DEVELOPMENT OF A MINIATURIZED ELECTROCHEMICAL PATHOGEN DETECTION PLATFORM FOR MICRO-TOTAL ANALYSIS SYSTEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Sylvia D. Kwakye

February 2010

DEVELOPMENT OF A MINIATURIZED ELECTROCHEMICAL PATHOGEN DETECTION PLATFORM FOR MICRO-TOTAL ANALYSIS SYSTEMS

Sylvia D. Kwakye, Ph.D.

Cornell University 2010

This thesis describes a translational research effort to design a portable device, the miniEC, that on its own, is useful for detection of nucleic acids of pathogens and other analytes, but can also be integrated into a complete micro-total analysis system. The design incorporated a microfabricated liposome-amplified biosensor, a potentiostat for electrochemical tests, a liquid crystal display, a pushbutton interface for user input, memory for data storage and a serial connection for networking the device to a PC.

In operation, the potentiostat powers the biosensor and is also responsible for amplifying and measuring the sensor's output. Specificity of the biosensor is ensured by DNA capture and reporter probes that hybridize with the target nucleic acid sequence. The reporter probes are coupled to liposomes entrapping the electrochemically active redox couple, potassium ferri/ferrohexacyanide. The capture probes are coupled to superparamagnetic beads. After hybridization, The liposome-target-bead complexes are captured by a magnet within the biosensor. The liposomes are then lysed to release the electrochemical markers unto an interdigitated ultramicroelectrode array. The redox activity of the markers on the electrodes is measured as a current, that is subsequently displayed and stored by the miniaturized instrumentation.

The portable miniEC system replaces electrodes, potentiostat and computer systems in conventional electrochemical set ups. Use of the inexpensive, low-power,

powerful and energy efficient MSP430 microcontroller together with other carefully selected, low power, off-the-shelf components, resulted in a design that can run for months on a single AA or AAA battery. Also, emphasis on low cost resulted in a compact design with a minimal number of components, that altogether, cost less than $50 in prototype quantities.

The accuracy of the miniEC was evaluated by comparing its performance to a standard bench-top electrochemical workstation in static and dynamic constant current amperometric experiments. In both sets of experiments, the inexpensive miniEC's performance was comparable in signal strength and sensitivity to that of the electrochemical workstation. It was then successfully used in the detection of dengue fever virus RNA. The design of the biosensor, hardware and embedded software was modular, general and flexible such that differentiated products could easily be generated for applications in different fields within a short development window. We provided an example of such a differentiated product with a step by step enhancement of the basic unit into a 5-channel multi-sensor unit. Applied to *Cryptosporidium parvum* detection, the multi-sensor miniEC was capable of single oocyst detection.

BIOGRAPHICAL SKETCH

Sylvia Kwakye grew up in Accra, Ghana, where she completed her elementary and secondary education. She obtained her Bachelor's degree in engineering with a concentration in computer science from Swarthmore College, Swarthmore, Pennsylvania. She worked as a research engineer in the Computational Biology Group at the Dupont Experimental Station, Wilmington Delaware. She received a MS degree in Biological Engineering from Cornell University in 2002 and stayed to pursue a Ph.D. Under the tutelage of Professor Antje Baeumner she has since conducted research into the development of a generic microfluidic biosensor system for the detection of pathogenic microorganisms.

To my family.

ACKNOWLEDGEMENTS

I would like to express my great appreciation to all those who have helped make my research experience so productive. For support and invaluable mentoring, I thank my advisor Professor Antje Baeumner. For scientific support also, I thank my minor advisors Professors Tom Bruce, Richard Durst and Peter Jackson.

I would also like to express my appreciation for the help of Mike Skvarla and the dedicated staff of the Cornell Nanofabrication Facility. I also thank all my lab-mates, for their help and usually useful suggestions. To Dr Richard Montagna (Innovative Biotechnologies International Inc., Grand Island, New York) thanks for sharing resources and expertise. For funding support I thank the Department of Biological and Environmental Engineering and the Cooperative State Research, Education and Extension Services (NYC-123404).

To my parents who have always loved, encouraged and sought to make my life better, Thank you. To my biggest cheer-leader, Stefan Minott, no number of "thank you"s could express how grateful I am for your constant support, encouragement and optimism. Thank you.

Last but not the least, I thank the faculty and staff of the department of Biological and Environmental Engineering for making this department a joy to be a part of.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACLK, Auxiliary clock

ADC, Analog-to-digital converter

AOAC,Association of analytical communities BOR, Brown-out reset

BSA, Bovine serum albumin

BSL, Bootstrap loader

CAN, Controller area network

CDC, Centers for Disease Control and Prevention

COTS,Commercial off-the-shelf

CPU, Central processing unit

DAC, Digital-to-analog converter

DCO, Digitally controlled oscillator

DC, direct current

DIP, Dual-in-line package

DMSO, Dimethyl sulfoxide

DPM, Digital panel meter

DNA, Deoxyribonucleic acid

DPPE, Dipalmitoyl phosphatidylethanolamine

DTD, Document type definition

EDTA, Ethylenediamine tetraacetic acid

ELISA, Enzyme-linked immunosorbent assay

FDA, Food and Drug Administration Fe^{2+}, Used to represent Potassium ferrihexacyanide

Fe^{3+}, Used to represent Potassium ferrohexacyanide

FLL, Frequency locked loop

HMDS, Hexamethyldisilazane

GIE, General interrupt enable

IC, Integrated circuit

I2C,Inter integrated circuit

I/O, Input/Output

IDUA, Interdigitated array of ultramicroelectrodes

LCD, Liquid crystal display

LED, Light emitting diode

LOCS, Lab-on-a-chip

LPM, Low-Power Mode

MCLK, Master clock

MCU, Microcontroller unit

MEMS, Microelectromechanical systems

MMCC, N-4-p-maleimidylmethyl cyclohexane-1-carbonyl

MMCC-DPPE, N-(4-(p-maleimidylmethyl)cyclohexane-1-carbonyl)-Diphosphatidyl palmitoylethanolamine

mRNA, messenger ribonucleic acid

MSSP, Master synchronous serial port

NASBA, Nucleic acid sequence-based amplification

NIST, National Institute of Standards and Technology

OG, Octyl-β-D-glucopyranoside

PBS, Phosphate buffered saline

PC, Personal computer

PCR, Polymerase chain reaction

PDA, Personal digital assistant

PDMS, Polydimethylsiloxane

PMMA, Polymethylmethacrylate

POR, Power-on reset

PSP, Parallel slave port

PUC, Power-up clear

PWM, Pulse width modulation

PWM, Pulse width modulator

op-amp, Operational amplifier

PCR, Polymerase chain reaction

RAM, Random access memory

RNA, Ribonucleic acid

RT, Reverse transcriptase

SARS, Severe acute respiratory syndrome

SATA, N-succinimidyl-S-acetylthioacetate

SCI, Serial communications interface

SMCLK, Sub-system master clock

SNR, Signal to noise ratio

SPI, Serial peripheral interface

SRB, Sulforhodamine B

SSC, Standard saline citrate

SSP, Synchronous serial port

μ-TAS, Micro-total analysis system

USDA, United States Department of Agriculture

WHO, World Health Organization

XML, Extensible markup language

CHAPTER 1

INTRODUCTION

The world-wide anxiety caused by the discovery, in May 2007, that a patient suffering from extensively drug-resistant tuberculosis (XDR TB) had traveled on commercial airlines, underscored the fact that, in a highly interconnected world, potential health threats need to be identified quickly.[1] Although the patient was known to have tuberculosis, he was not considered contagious and officials at the US Centers for Disease Control and Prevention (CDC) did not know he had the very dangerous drug resistant strain, until test results came in several weeks after the initial diagnosis. The patient was immediately quarantined and treated when he returned to his home in the United States of America. Airline passengers and crew who had flown with him were tested for the disease. One important implication of this event is that, in poorer countries where healthcare is resource-limited, a case like this can easily cause a world-wide epidemic. In fact, this is what had happened earlier in 2002-2003 with the sudden acute respiratory syndrome (SARS) epidemic.

The SARS outbreak begun in 2002 in the rural Guangdong province of China, went mostly unnoticed, and spread rapidly to other countries via air travel.[2] The epidemic reached the public spotlight in February-April 2003 when two Americans, one a business man who had traveled to the region and the other an English teacher at Shenzhen Polytechnic died of SARS. Recognizing the threat, the World Health Organization (WHO) created a network of 13 laboratories in 10 countries. This team identified the coronavirus associated with SARS in 2 weeks and had its entire genome sequenced in another 2 weeks.[3] WHO was then able to issue guidelines to health workers and help governments mount educational campaigns that helped to greatly reduce the transmission rates of the disease.

WHO is currently monitoring several outbreaks of disease that include Ebola haemorrhagic fever in Uganda, Rift Valley Fever in Sudan and the relatively new threat of the avian influenza virus H5N1, that has evolved to infect humans, in several Asian countries. Thanks to an extensive education campaign and containment actions of WHO and the affected states, the case fatality rate of H5N1 decreased from 73% in 2004 when it first reached the spotlight, to 43% in 2005.[4] Since 2005 however, the case fatality has increased as world attention has waned and migrating birds have spread the disease to other geographic areas.

For many tropical diseases that affect developing countries and evolving diseases like Ebola and H5N1, there are no specific treatments or effective human vaccines. The only way to reduce such infections and death is to raise awareness of the risk factors and the protective measures individuals can take to prevent exposure. Health workers in these endemic regions clearly need access to sophisticated diagnostic tools for research and rapid detection of pathogens so they can offer effective education/treatment. Unfortunately, the best diagnostic technology often require resources such as stable electricity supplies, air-conditioned laboratories, highly trained personnel and rapid transportation of samples that may not yet be available in developing countries.[5] When these conditions exist, care may be out of reach for many people because of the expense of treatment and travel to the location of the medical facility. In developed countries, by contrast, care may be readily available but as the TB case showed, it is sometimes not enough to have a lab test for a pathogen if there is a significant delay in obtaining the results. Ideally, the test results should be available at the point-of-care when an initial diagnosis is made. Furthermore, the availability of accessible and affordable healthcare is a growing challenge in many industrialized countries. The problem is more acute in the United States where healthcare is not publicly funded. In spite of this need,

many USA hospitals are closing because their operating costs make them unsustainable.[6,7] For those that remain open, the United State's Agency for Healthcare Research and Quality estimates that, a third of the hospitals that provide services to uninsured patients operate with a negative total income margin.[8]

A clear solution to the challenges of support infrastructure deficits, slow diagnostic processes or cost reduction pressures is the development of inexpensive, portable tools for point-of-care diagnostics. In both high and low-resource locales, the tools would require simplicity, specificity, accuracy and reliability, as well as low costs of device acquisition, operation and maintenance. In addition, low resource locales require freedom from the need for grid power and reliability in extreme conditions.

1.1 A Growing Need

The 2006 North American *E. coli* outbreak, made headlines around the world because the rapid onset and widely distributed nature of the outbreak, raised concerns about the vulnerability of centralized food production to bioterrorist attacks. In this case, leafy greens grown in California were contaminated by migrating cattle manure containing *E. coli O157:H7*, a toxigenic strain of *E. coli*. The greens, bagged and shipped to consumers across the United States and Canada, caused sickness, and in some cases death in those who consumed the bagged salads.[9] To prevent future outbreaks, the USDA is considering a variety of new leafy green regulations from pathogen lab tests at every harvest, irradiation of prepackaged fresh greens, to sterilization of fields so that no live organisms remain in the soil.[10] Many of the proposed regulations have been opposed by farmers because they are considered impractical, unsustainable or too expensive.[11,12] An accurate, rapid, specific and inexpensive test that can be administered at any point in the distri-

bution chain from harvest to packaging will help the industry keep their products safe for consumption at reasonable cost. This need is further underscored by the numerous cases of food borne illness that do not make the headlines. According to the Centers for Disease Control and Prevention, 76 million Americans suffer from food-borne illnesses every year.[13] Of these, 325,000 are hospitalized and more than 5,000 die. Furthermore, several important diseases of unknown cause have turned out to be complications of foodborne infections. Examples include the Guillain-Barre syndrome (acute peripheral nerve inflammation), caused by *Campylobacter* infection,[14] and hemolytic uremic syndrome (acute kidney failure), caused by infection with *E. coli O157:H7* and related bacteria.[15]

The need for inexpensive diagnostic tools is not thus not confined to the area of health care. Such tools will also be useful in food safety testing and a variety of applications from environmental monitoring to preventing bioterrorism or in the event of an attack, limiting mass casualties.

Environmental monitoring is important for evaluating and mitigating threats to the environment and public health, tracking natural resources and for reducing the costs associated with waste treatment. Current monitoring methods required by regulatory bodies such as the United States Environmental Protection Agency are often expensive, time-consuming and require skilled personnel and a laboratory equipped with expensive analytical tools. In some large scale projects such as the cleanup of Super Fund sites in the USA for instance, laboratory analysis can account for 80% of the cost of the remediation effort.[16] In addition to the costs, the integrity of the analyses can also be compromised at any point within the usually multiple day process of sample collection, storage, transport and analysis. Portable instrumentation that allows on-site sampling and analysis has the potential to make environmental monitoring simpler, faster and inexpensive. The instruments can be

designed to work unattended and to store data for periodic retrieval. Devices with wireless data transmission capability will enable more routine monitoring of remote or hostile environments where prolonged human activity is disruptive, limited by distance or limited by life threatening conditions. These field portable devices installed for environmental monitoring could also incorporate intrusion sensors to detect intentional damage such as might occur in a biological or chemical attack. Early detection of the biowarfare or chemical agents is crucial for limiting the impact of an attack. Furthermore, the resources and diagnostic tools required for rapid identification of pathogens in the event of a bioterrorist attack are the same as those required for monitoring populations during a natural disease outbreak. Thus the tools developed for low-cost point-of-care diagnostics will be directly useful in equipping first responder teams to promptly identify biowarfare agents and to coordinate a rapid response for treatment, quarantine and vaccination or evacuation of affected and threatened persons.

1.2 Technologies For Decentralizing The Laboratory

Today, the convergence of three technologies - genomics, microelectronics, and microfluidics gives us the ability to shrink the analytical lab into portable battery operated systems for clinical work in resource-constrained locales and to provide distributed diagnostics in agricultural, environmental and food safety monitoring applications.

Genomics is the large scale study of all nucleotide sequences of an organism. Extensive governmental and private sector research efforts in this arena have made available huge databases of the genomic makeup of thousands of organisms. The best known of these repositories is GenBank, that as of January 2007, contained nucleotide sequences for more than 240,000 named organisms.[17] This information

is invaluable for the development of nucleic acid based detection assays that are highly specific. Furthermore, deeper understanding of the structure and function of nucleic acids has also given rise to a number of nucleic acid amplification schemes such as the polymerase chain reaction (PCR) and nucleic acid sequence based amplification (NASBA), that make such assays highly sensitive as well.

Microelectronics, the most mature of the three technologies, relates to the study and manufacture of small electronic components in the micron and sub-micron range. Advances in fabrication methods, device structures, and materials have led to computers shrinking from the 1800 square feet ENIAC to a device that can be carried in a pocket. The impetus for the amazing miniaturization breakthroughs in microelectronics is economics. Scaling down solid-state devices reduces cost but improves performance and power, giving any company with the latest technology a large competitive advantage in the marketplace.[18]

Microfluidics borrows greatly from the fabrication methods first developed for microelectronic systems. Microfluidics is the branch of micro machines (also called microelectromechanical systems or MEMS) that deals with the development of miniature devices that move, mix, control and react fluids or solutions containing dissolved or suspended species. Processing samples of any size in a bioanalytical system involves moving, mixing and reacting reagents. Thus, a major component of shrinking the analytical lab is the miniaturization of fluid handling systems. Here, microfluidics offer obvious advantages in the reduced consumption of reagents; faster and more sensitive reactions due to enhanced effects of processes such as diffusion and mass transport; increased throughput through parallel processing; and reduced expenses in terms of power and reagent consumption.[19]

There is an interesting synergy between the three technologies. Technological advances in microelectronics, particularly in the production of central processing

units (CPUs) and memory chips, provide better instrumentation for measurement and analysis in genomics. Microfluidics benefits from advances in microelectronics since both fields share many fabrication techniques. Microfluidics also enables large numbers of genomic assays to be run simultaneously. Recently, as engineers recognize the limits of silicon microelectronics, some are beginning to consider DNA computing.[20–22] Genomics contributes to the synergy by providing the tools for such considerations. Ultimately, this synergy is sure to produce more powerful and more cost effective new utilities for shrinking the analytical lab.

1.3 Micro-Total Analysis Systems

When microfluidic systems are integrated with electronic circuitry, biological target recognition elements (e.g. a biosensor) and other micro-mechanical devices for enhanced fluid control and sensing, they are described as micro-total analysis systems (μTAS) or lab-on-a-chip systems (LOCS). In principle, μTAS perform all analytical functions for sample pretreatment; amplification, concentration or dilution; mixing or separation; biochemical reactions and detection with minimal manual intervention until analysis is complete.[23] By this definition, the first μTAS was a gas chromatograph reported by researchers at Stanford University in 1979.[24] Most literature however put the start of the μTAS revolution in bioanalytical systems in the early 1990s when Manz published a seminal paper on the topic.[23] This period also coincided with the beginning of the Human Genome Project which subsequently benefitted from the availability of new analysis tools. Gene expression analysis for instance was miniaturized unto microarrays enabling the monitoring and comparison of thousands of genes at once. There were also remarkable successes achieved in reducing analysis times in capillary electrophoresis and chromatography where the reduced column dimensions greatly increased

speed and separation efficiency.[25] The pharmaceutical industry in particular has also benefitted greatly from the use of lab-based instrumentation that use microfluidic chips for high throughput screening for drug discovery.

A review of the literature on the current status of μTAS shows that almost all major recombinant nucleic acid techniques and clinically important protocols such as electroporation[26] and the polymerase chain reaction[27,28] now have microfluidic versions. In addition, it has been a useful platform for studying new physical phenomena on the micron and sub-micron scale. To date, the numerous research articles and patents on μTAS have resulted in very few commercial products. Many research groups, including ours, are faced with numerous technical challenges including representative sample preparation, design of functional miniaturized pumping systems and the design of practical micro-macro interfaces to connect the miniaturized device to the rest of the world. These challenges among others have slowed the pace of product development. In addition to the technical challenges, there is the difficult missionary task of attracting investors to fund novel device development and subsequently marketing the new technology. Examples of the few μTAS commercially available include Sandia's MicroChemLab biotoxin detector[29] and Abbott Laboratory's i-STAT portable analyzer.[30] The ABO card produced by Micronics Inc. (Redmond, WA) is particularly notable for the simple elegant design.

The MicroChemLab comes in two versions, one for gas sensing and the other for liquid sensing. The core sensor components of the device miniaturize the operations of a gas chromatograph and a mass spectrometer respectively. In the gas detector, a pre-concentrator pad absorbs gases that are subsequently released when the pad is heated. The desorbed gases enter the separation column of a miniaturized gas chromatograph. The chemicals in the gases are identified by a surface-acoustic-

wave sensor as they are eluted from the column.[31] The liquid MicroChemLab can detect pathogens via DNA analysis and toxins, via their protein signatures.[31] Fluid handling in the microchannels is achieved with high voltage electrokinetic forces while sample preconcentration and injections occur through manipulation of electrophoretic fields. For both sensors, sample analysis time is less than 5 minutes, with high specificity and sensitivity in the range of 10 to 100 parts per billion for chemicals and picomoles for biotoxins.[31] Stability, however is only in hours since drift is a significant issue. Each MicrochemLab is about the size of a small suitcase and weighs 25 pounds when packaged with sample collection units and supporting electronics.[32] Thus while portable, the devices are not suitable for extended handheld use but are very well suited for unobstrusive environmental monitoring in a manner analogous to a smoke detector. The gas-phase MicroChemLab is in field use in the Boston subway transit system (Massachusettes, USA) where it continuously tests the air for biotoxins. The liquid sensor is also being tested as a tool for monitoring water systems in other parts of the United States. The current cost of a MicroChemLab is on the order of $10,000 to $20,000. The researchers hope that further design improvements and more widespread adoption will help reduce the price.

The i-STAT performs a wide selection of blood tests using disposable microfludic cartridges and a microelectronic analyzer. Each cartridge contains a system of microchannels, a reagent pouch, a microfabricated electrochemical sensor, a port for sample injection and a chamber to contain waste. To run a test, a whole blood sample of approximately 1 to 3 drops is dispensed into the sample port and then inserted into the analyzer. Results are typically obtained within 2 minutes. Over 20 different kinds of cartridges are available to perform a number of tests for blood gases, electrolytes, coagulation, hematology and cardiac markers. The i-STAT sys-

Table 1.1: I-STAT Changes the Economics of the Market[33]

	Traditional cost profile	i-STAT cost profile
Labor	4	0.50
Equipment	0.75	0.30
Reagents/Cartridge	0.75	3.50
Total	5.50	4.30

tem features all the essential requirements for point-of-care diagnostic testing such as portability, freedom from an electric grid, ease-of-use and short time-to-result. As one of the first of such devices on the market however, it is also expensive. The analyzer retails for about $5,000 - $8,000 depending on the assessories bundled with it. The cartridges range in price from $1.75 for a simple glucose test to $20 for cardiac marker Troponin I.

Figure 1.1: The i-STAT blood chemistry analyzer.

The performance of the i-STAT system has been verified by numerous clinical studies, particularly in the area of pediatric care, where the small sample size used by the i-STAT is an attractive capability.[34–37] At current prices however, most of the studies conclude that the per test cost of supplies to run a test are comparable to traditional centralized lab costs. The savings are primarily in time, labor and fixed overhead costs to house and maintain laboratory equipment. Table 1.3, which shows the cost profile for the i-STAT from the manufacturer's literature, acknowledges this fact. An example of this cost profile is reported in a three

month study by Macnab et. al that monitored the blood gas levels of critically ill pediatric patients prior to inter-hospital transfers at British Columbia Medical School. The researchers found that the reduction in time taken for stablizing the patient, 105 versus 526 minutes, accounted for all their cost savings.[37] The time reduction directly reduced the cost of aircraft charter charges, technician call-backs and paramedic personnel labor costs.

Figure 1.2: The plastic microfluidic cartridge of the Micronics ABO blood typing card.[38]

Unlike the i-STAT and MicroChemLab, Micronics' ABO card has no reader and its operation does not require a power supply. It is about the size of a credit card (Figures 1.2 and 1.3) and processes a droplet of whole blood to identify blood type within 30 seconds. All reagents are pre-loaded in the 3 wells at the top of the card. To use, a drop of blood is put into a fourth well. The capillary effect starts the blood flowing through the microchannels. The user then pushes a small on-card bellows located over the large waste well at the bottom of the cartridge. This draws the reagents and sample together. In real time, a viewing windows shows whether or not agglutination occurs. The cartridge is packaged within a simple

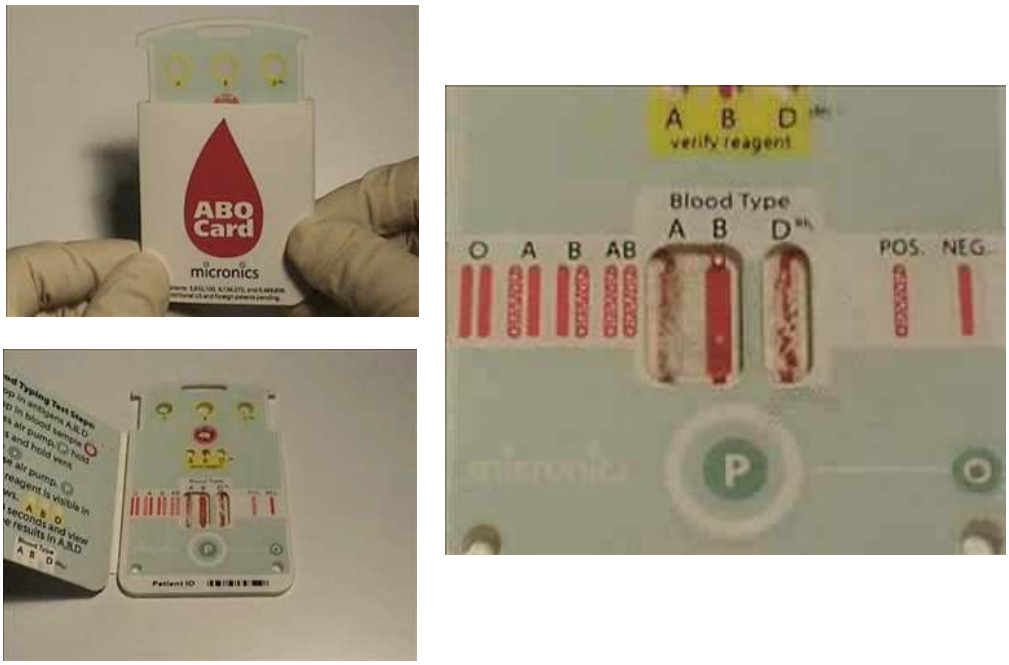


Figure 1.3: The Micronics ABO card for blood typing in convenient packaging, with printed instructions for running a test and a simple reference panel, to help determine the blood type.[38]

card housing, printed with instructions for running a test and a simple reference table to help the user determine the blood type. All waste is stored on card. A second generation version of the card in development prints the reagents directly in the microchannels.

Micronics Inc. is also actively involved in the development of the DxBox, the most ambitious μTAS project reported to date. It is funded by the Bill and Melinda Gates Foundation Grand Challenges in Global Health initiative, with the aim of producing a portable diagnostic system for healthcare in the developing countries. The DxBox is an interdisciplinary effort incorporating microfluidic innovations from Micronics; diagnostic assays developed by teams at the University of Washington and Nanogen Inc.; instrument design by Invetech; and finally, systems integration, testing and validation by the Program for Appropriate Technology in Health Organization. Successes to date include the design of a self-

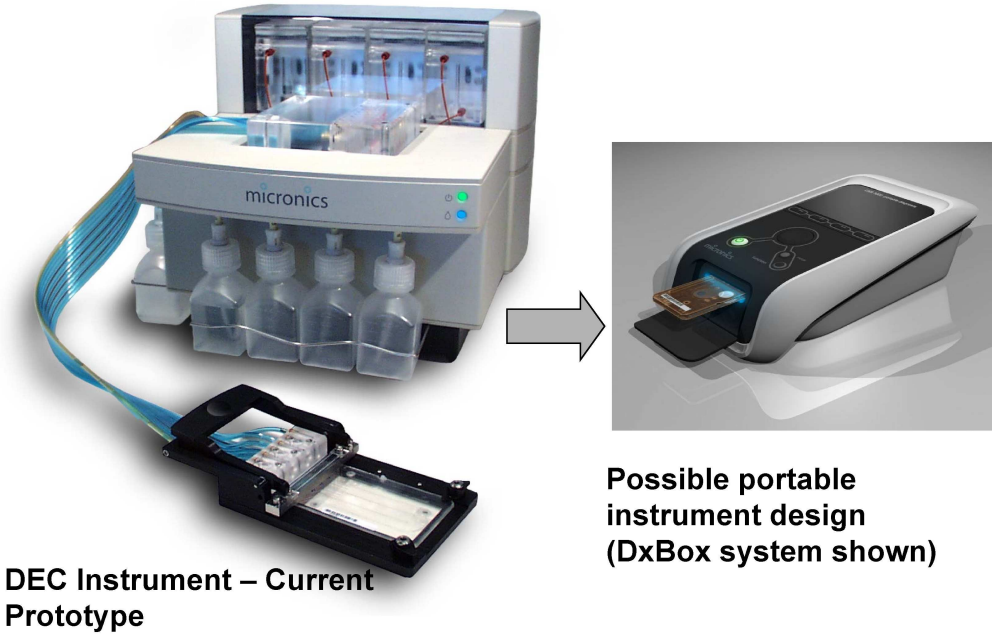


Figure 1.4: The current diagnostic workstation and a proposed portable version.[39]

contained microfluidic card capable of performing nucleic acid amplification and an immunoassay for a target pathogen.

1.3.1 Hybrid Micro-Total Analysis Systems

Hybrid analysis systems that recycle or repurpose existing commercial products such as laptops and PDAs, for incorporation into portable analytical systems are much further along in terms of products available commercially than systems that are built from scratch along wholly μTAS principles. An example is the electronic nose reported by Kim et al.[40] The sensor module was designed to fit into the slot for a memory card of an iPAQ 5550 personal digital assistant (PDA). The PDA was then programmed to handle data acquisition, storage and signal processing. The system benefits from having components like serial communication, a sophisticated LCD and keypad already built-in. Another example of a hybrid micro-analytical system is DiagnoSwiss' pairing of their Gravi-Chip with PalmSens' pocket-PC

based electrochemical analyzer. The GRAVI-Chip is a microfluidic biosensor that uses a standard enzyme-linked immunosorbent assay (ELISA) to detect the presence of target antigens in solution. While traditional ELISA typically uses optical reporters, the GRAVI-chip incorporates an electrochemical reporter that undergoes redox reactions on microelectrodes embedded into the microfluidic channels. Here also, the pocket-PC handles the data acquisition and control of the Gravi-chip. The PalmSens system may be replaced by specialized software running on a generic laptop computer.

Recognizing the potential of portable analytical systems, a few electronics companies have begun working with researchers to produce portable hybrid analytical systems. Motorola, a manufacturer of cell phones, for instance, has in collaboration with Gentag Inc. patented the concept of using cell phones as a universal reader for disposable wireless sensors.[41] A second patent,[42] owned by Gentag, covers the use of cell phones in combination with removable sensor modules.

1.4 The Research Problem

Most of the verified μTAS systems currently on the market such as the i-STAT are still too expensive for the low resource communities that have great need for such devices. The main goal of this research is to help mitigate the cost barriers for developing such devices with the design of a platform that will integrate a nucleic acid biosensor, hardware, and software to serve as the basis for the development of field portable, consumer friendly, inexpensive micro-total analysis systems.

1.5 Background

Before discussing the details of the application framework of the portable detection system in subsequent chapters, it is useful to define some of the terms associated with nucleic acid hybridization based biosensors, review background techniques, preliminary work and a number of important research contributions that implement detection schemes similar in concept to this research project.

1.5.1 Nucleic acids – definitions

A single strand of nucleic acid is built up from a set of four bases each of which is connected through a sugar molecule to a phosphate backbone. The base-sugar-phosphate unit is called a nucleotide. The sugar in a nucleotide is either a ribose or deoxyribose sugar hence the names ribonucleic acid (RNA) and deoxyribonucleic acid (DNA). DNA resides primarily in the nucleus of an organism's cell while RNA can be found throughout the cell. DNA is the template for making RNA which in turn is the template, factory and transport entity for proteins. RNA thus has several types depending on its location and function in a cell. Messenger RNA or mRNA for instance, is a short-lived molecule with the function to move the information contained in DNA for the synthesis of specific proteins from the nucleus to the protein synthesis sites (ribosomes) in the cytoplasm of a cell. Two other types of RNA that help in protein synthesis are ribosomal RNA (rRNA) and transfer RNA (tRNA). The rRNA is a component of the ribosomes while tRNAs transport amino acid units needed for protein synthesis from the cytoplasm to the ribosomes.

In its natural state, DNA is a double-stranded helical molecule while RNA is typically single-stranded but folded into structured shapes with loops and hairpins. The nucleotides within a strand are linked by the phosphate group in each

nucleotide connecting the 5' (5 prime) carbon atom of its sugar molecule to the 3' (3 prime) carbon atom of the adjacent nucleotide's sugar molecule. Consequently, the nucleotide at one end of a strand will have a free 5' carbon on its sugar molecule and that at the other end will have a free 3' carbon. This polarity is important as nucleic acid sequences hybridize in such a way that the 5' end of one strand hybridizes with the 3' end of another strand.

Formation of double stranded DNA and nucleic acid hybridizations rely on the formation of hydrogen bonds between complementary bases on each strand of nucleic acid. Three of the bases, adenine (A), cytosine (T) and guanine (G) are common to both DNA and RNA. The fourth base in DNA is thymine (T) and the corresponding base in RNA is uracil (U). The base pairing are such that cytosine only forms bonds with guanine while adenine will only form bonds with thymine or in the case of RNA, uracil. The composition of nucleotides varies greatly between different organisms.

In a nucleic acid assay, the target refers to the specific nucleic acid sequence that is unique to the organism the assay is designed to detect. Probes are short sequences synthesized so that they are complementary to sections of the target sequence. Two kinds of probes - capture and reporter probes - may be used in a typical sandwich assay. A capture probe is attached to a solid substrate so that when it binds to the target sequence, the target is effectively anchored to the substrate. The reporter probe is attached to a signal generating particle referred to as the reporter. The reporter enables visualization and/or quantification of hybridization. Examples of reporters include fluorescent molecules such as sulphorhodamine B (red) and carboxyfluoresceine (green),[43] chemiluminscent compounds like Origen phosphoramidite,[44] phosphorescent particles such as Yttrium oxysulfide[45] and sol particles such as quantum dots,[46] colloidal gold[47–49] and liposomes.[50,51]

A nucleic acid assay may use one or both types of probes as illustrated by Figure 1.5. When just one probe is used, hybridization is described as direct. In this case, the sample is pre-treated to label the target with a reporter and it hybridizes only once to a capture probe (Figure 1.5a). Alternatively, the target may be attached directly to a substrate and hybridizes to a reporter probe (Figure 1.5b).

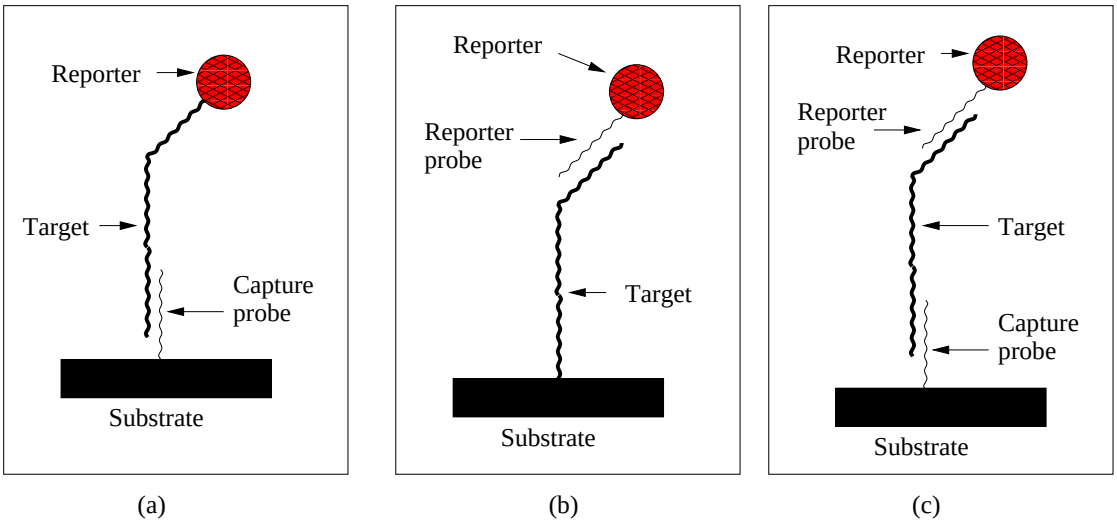


Figure 1.5: Diagram showing the components of (a), (b) direct hybridization and (c) sandwich hybridization

Assays in which both reporter and capture probes are used to identify the target are referred to as sandwich hybridizations because of the formation of capture probe-target-reporter probe complexes (Figure 1.5c). The additional hybridization adds another degree of specificity making the sandwich hybridization less prone to non-specific binding interference than direct hybridization.

1.5.2 Nucleic acids – amplification

Target nucleic acids in many clinical and environmental samples are often present in such minute quantities that direct detection is difficult, especially if single cell analysis per mL (or even per hundreds of liters in environmental analyses) is de-

sired. Most nucleic acid detection assays as a result have an amplification step. Currently the most used and well-known is the polymerase chain reaction (PCR). PCR exploits the ability of DNA polymerase to synthesize new strands of DNA complementary to template DNA from a pair of oligonucleotide primers.

PCR starts with double stranded DNA that is heated until it separates (denatures) and then cooled so that the primers can anneal to the single strands of DNA. DNA polymerase extends the primers and results in 2 copies of the original DNA. The new copies in turn serve as templates in another heat-cool cycle to synthesize more DNA. After n cycles, there would theoretically be 2n copies of the DNA. PCR is very sensitive and specific and as such has been extensively used in molecular biology and clinical diagnosis.[52] Due to its importance in clinical diagnostics, several groups viewed it as a target for miniaturization and have reported successful outcomes.[27,53]

PCR was designed to amplify DNA. For detection of pathogens however, mRNA is the preferred target as its presence is a better indication of viability than DNA.[54–56] Compared to DNA and the other types of RNA, mRNA molecules are very short-lived. They are rapidly degraded by enzymes that are very stable under most conditions.[57–59] Most of a non-viable cell's mRNA is degraded within minutes of cell death. DNA on the other hand can persist for hundreds of years after the organism is dead.[60,61]

Reverse transcriptase PCR or RT-PCR is a variant technique of PCR that amplifies RNA. It involves an initial step that uses reverse transcriptase to convert RNA to DNA before proceeding with PCR as described. The RT-PCR procedure requires that the sample be completely free of DNA before the procedure starts so it is not amplified along with the RNA. Researchers typically accomplish this step by first treating the sample with enzymes (DNase) that degrade DNA. This

treatment is not always effective at removing all of the DNA and as such raise the probability of false positives in subsequent detection assays.[62,63] Nucleic acid sequence based amplification (NASBA) is a relatively new isothermal scheme that amplifies just RNA.[64]

NASBA is more interesting to us than the more widely used PCR and its variant RT-PCR for several reasons. NASBA was specifically designed to work with RNA and is optimized for that purpose, whereas PCR as previously mentioned was designed to amplify DNA. NASBA does not require thermocycling and as a result is much simpler to realize on a chip. It is also more productive when targets sequences are short.[65] Furthermore, since RNA molecules are present in higher copy numbers than the expressed gene or DNA, NASBA is more sensitive for this application.[66] PCR doubles the number of nucleic acid with each cycle. NASBA, however, generates 10–100 copies of RNA in a cycle so it is at least five times faster than PCR.[65] More important though, is the fact that RNA conclusively signals a viable organism while DNA is present in dead organisms for a long period of time. Since NASBA only amplifies RNA molecules, a biosensor can be designed to specifically detect only viable organisms. If PCR was used, traces of genomic DNA of dead organisms could easily produce false positive signals.

1.5.3 Nucleic acid sequence based amplification (NASBA)

The NASBA procedure involves three sequential steps catalyzed by three enzymes, reverse transcriptase (RT), Ribonuclease H (RnaseH), T7 RNA polymerase and two target specific primers to amplify the single stranded target sequence. Using a single stranded RNA molecule as substrate, it produces isothermally 10^9 – 10^{12} copies of the anti-sense RNA strand within 90 minutes (Figure 1.6).

The first step in NASBA converts input RNA into an RNA-DNA hybrid molecule

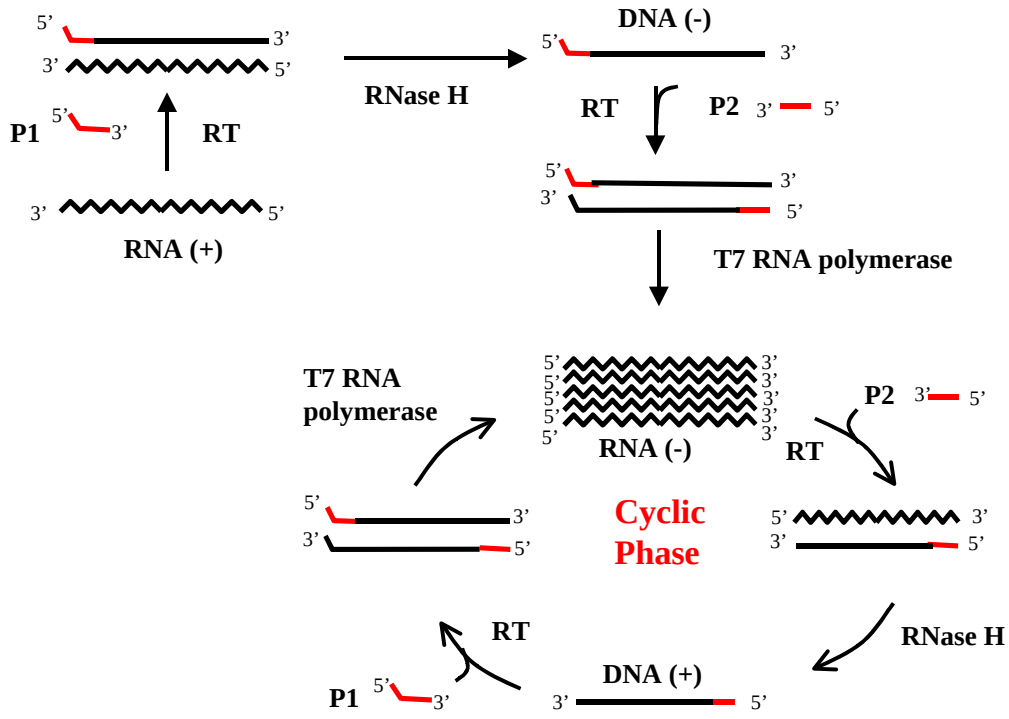


Figure 1.6: Schematic of the nucleic acid sequence based amplification (NASBA) reaction.

using the reverse transcriptase (RT) enzyme and a primer (P1) specific to the target organism. P1 also contains the promoter sequence for T7 polymerase. The RNA strand in the hybrid molecule is degraded enzymatically by RNaseH which leaves a single strand of DNA. Employing a second primer (P2) and RT, a double stranded DNA is synthesized from the single strand bearing a complete T7 promoter sequence. From this DNA molecule, antisense single-stranded RNA molecules are produced via transcription using T7 RNA polymerase. These RNA molecules are subsequently used as templates for the synthesis of double stranded DNA and thus further transcription (cyclic phase). A typical NASBA reaction requires an initial temperature of 65 ^{o}C (5 minutes) for the first primer annealing step after which amplification occurs at a constant 41 ^{o}C and delivers about 10^{9}-fold amplification in 90 minutes.[65] Considering the micron-sized microchannels and close proximity

of all reacting species, it is estimated that a 5 to 10 minute NASBA reaction will produce enough target sequence for detection in a microfluidic chip.

1.5.4 Membrane-strip biosensors

The current standard for portable, rapid diagnostic tests are the dipstick tests, referred to in scientific parlance as lateral flow assays (LFAs) or capillary-migration assays. LFAs refer to tests in which samples in solution with other reagents are carried to a detection zone on a porous strip of membrane by capillary action from an application zone. Glad and Grub are credited with the development of lateral flow assays.[67,68] Their method involved coating a strip of membrane (polyvinyl chloride or cellulose acetate) with antibodies for a particular antigen, drying the membrane and then placing one end of the membrane in the sample solution containing the antigen to be quantified. After the solution had migrated up the membrane via capillary action, the membrane strip was removed and incubated in a solution of fluorescein-labeled antibodies. The area where fluorescein bound on the strip was an indication of how high the antigen had migrated and therefore of how much antigen was in the sample solution.

Today, the most common strip material is nitrocellulose and the assays are simpler. Figure 1.7 is a schematic representation of a lateral flow membrane strip assay. Typically, a capture molecule that binds specifically to the target of interest is immobilized in a line on the surface of a membrane strip. The sample pad area of the strip is then placed in a solution which is a mixture of the sample solution and a solution containing a reporter molecule conjugated with a visually apparent particle in a running buffer that facilitates fluid flow through the strip. In some applications, the particle congugate is pre-deposited on the strip. The reporter molecules bind to the target molecule if it is present in the sample solution. As the

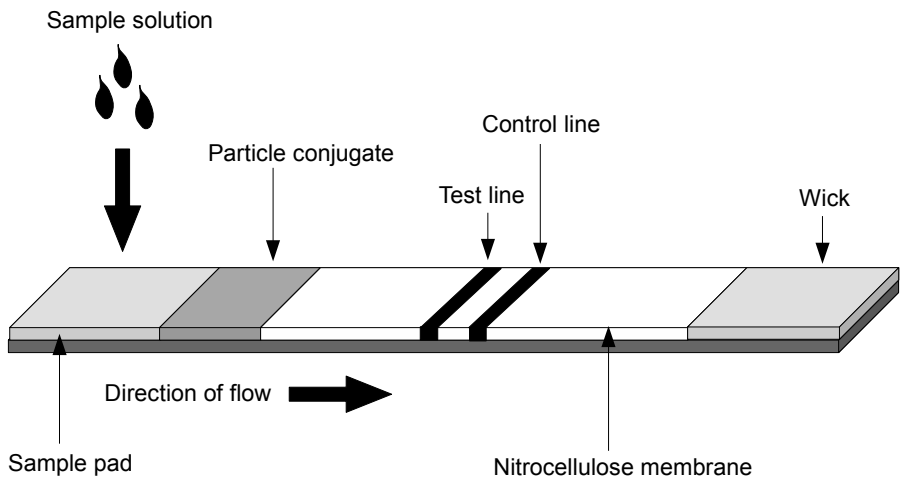


Figure 1.7: A typical lateral flow assay format. The sample solution is applied to the sample pad and flows towards the wicking pad. If the target molecule is in solution, it binds to the particle congugate on the conjugate pad. At the test line, immobilized capture molecules bind the target-particle conjugate to produce a visible line which denotes a positive test result. A visible control line confirms that the sampled flowed the length of the test strip.

solution migrates up the strip, the target-reporter complex binds to immobilized capture molecules and appears as a solid line. Some strip assays have a second line of immobilized probes to capture excess reporter molecules. This second line serves as a control to indicate that the strip test flowed correctly. Finally, most commercially available LFA products contain a wicking pad that absorbs excess solution to keep wastes contained. LFAs are suitable for a wide range of sample types including serum, saliva, blood and urine.[69] With extra sample preparation steps, they can also be used with tissue and cultured cells.[70] Numerous dipstick assays can be found in food safety testing,[70–73] environmental monitoring[74,75] and point-of-care clinical applications.

Dipstick assays in point-of-care diagnostics enjoy considerable commercial success due to robustness, simplicity and relatively low cost per test. Perhaps the most widespread test of this kind is the kit for pregnancy tests. Depending on the brand, pregnancy test kits may retail for as little as $1 per test. Figure 1.8 shows

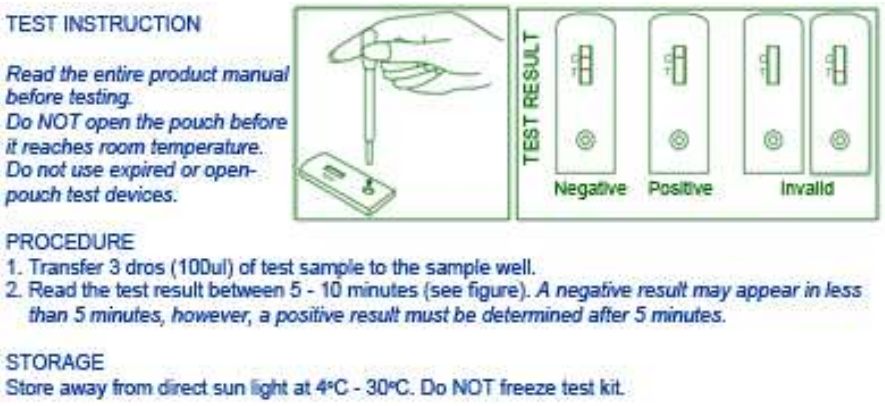


Figure 1.8: Instruction insert accompanying an Acro Biotech (Rancho Cucamonga, CA) dipstick test.

instructions from a commercially available strip housed within a plastic cartridge that illustrates the simplicity and ease of use of the typical dipstick product.

Diagnostic tests have been developed, tested and distributed for socioeconomically devastating diseases such as Schistosomiasis,[76,77] AIDS[78–80] and malaria.[81–83] Diagnostic assays are also available for chronic diseases like chagas' disease[84,85] and leprosy[86] that are not easily identified by symptoms in their early stages. Dipstick assays have also proven useful in epidemiological studies of disease outbreak.[87,88] In one such study, Dash and co-workers were able to determine that the increased incidence of life threatening dengue hemorrhagic fever and dengue shock syndrome in India, was due to dengue fever virus serotype-3 replacing the less virulent serotype-2.[88]

Lateral flow assays have traditionally been used with antigens and antibodies as the capture and reporter molecules because of the high selectivity of the antibodyantigen reaction. The Baeumner group (that I am a part of) however has successfully developed nucleic acid based LFA. Using liposomes as the reporter particle, our laboratory researchers have been able to rapidly detect numerous pathogens including E. coli,[89] *Cryptosporidium parvum*,[90] and *B. anthracis*.[91] Furthermore, significant contributions have been made to the body of methods

developed for lateral flow assays. One such contribution is the use of a universal membrane and universal liposomes as the reporter.[90] The universal membrane strip has a line of streptavidin immobilized in the detection zone instead of a target specific capture probe. Likewise, the universal liposomes are not specific to a target but bear a generic oligonucleotide sequence. The user makes the strip specific at the time of use with reporter and capture probes designed to bind not only to the target sequence but also to the generic sequence on the liposomes and streptavidin respectively. The 3' end of the reporter probe is complementary to the generic liposomal oligonucleotide, while the 5' end is complementary to the target sequence. The capture probes are complementary to the target sequence at the 3' end and biotinylated at the 5' end to form strong biotin-streptavidin bonds that effectively immobilize the target. A similar universal system with the biotin-streptavidin bond occurring between the liposome and the reporter probe instead was successfully demonstrated as well.[92] Another notable contribution is a novel lateral flow assay, demonstrated with the detection of *Streptococcus pyogenes*, that does not require an amplification step but instead uses multiple reporter and capture probes per target sequence.[93] This method reduces the cost per test by avoiding costly enzymatic gene amplification reactions. Selection of the capture and reporter probes was automated by a computer program designed to complement the assay. The probes were selected with no iterative optimization or re-design of the oligonucleotides, making this method an excellent platform for rapid biosensor design.

LFA technology has a number of limitations that may be resolved by μTAS technology. The most obvious is the subjective interpretation of results. Unless the strip comes with a "reader" to interpret the results, the assay is only qualitative and subject to reader variation when the signal is faint. Furthermore, results may

be invalid if not interpreted within a certain time frame, so, it may be difficult to compare archival strips. There is also limited capacity for testing multiple samples and analytes in parallel or for automation of sample preparation steps. Finally, in its most commonly used format, with antibodies and antigens, false positives are more likely for diseases such as malaria where antigen may persist in measurable quantities long after the cure of infection.[94]

1.5.5 Optical microfluidic biosensor system

An optical detection scheme with a bench-top microscope was first used to prove the concept of a microfluidic biosensor for the proposed application. Design, testing and decisions about the substrate, geometry and fabrication of the biosensor and subsequent successful implementation of the capture and detection modules in this preliminary research resulted in a generic semi-disposable microfluidic biosensor for highly sensitive detection of pathogens via their nucleic acid sequences.[95,96]

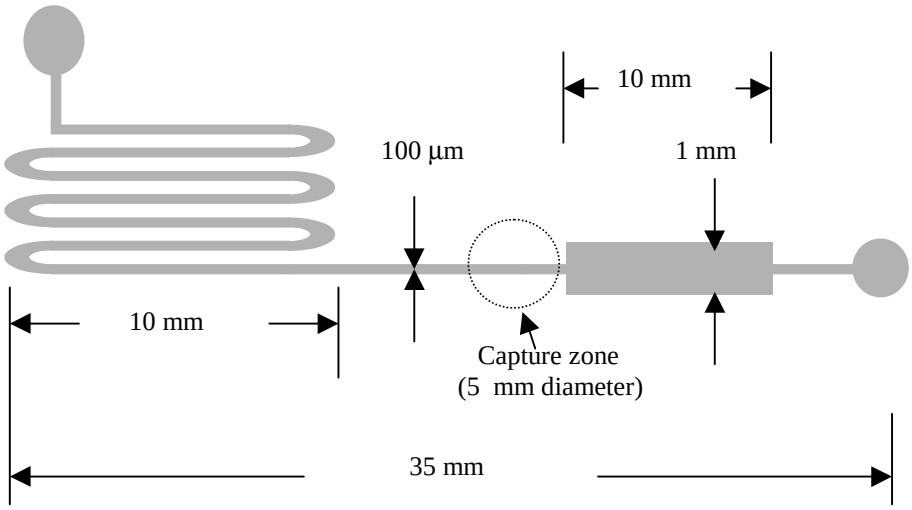


Figure 1.9: Layout and dimensions of the microfluidic biosensor.

The biosensor consists of microchannels with defined areas for capture and detection of target pathogen RNA sequence fabricated in polydimethylsiloxane

(PDMS). The PDMS devices were mounted onto a reusable polymethylmethacrylate (PMMA) stand for biosensor operation. The configuration of the biosensor (Figure 1.9) was the result of several iterations of microchannel design. The enlarged 1 mm by 10 mm area in the layout is designated the detection zone. The 5 mm microchannel length preceding the detection area is the capture zone. A permanent magnet is embedded in the PMMA stand below the microchannel in this area to provide a magnetic field that will hold magnetic beads in solution in place. Fluid flow into the microchannels was initiated with vacuum suction provided by a 10 – 20 mL syringe connected by tubing and plastic pipe fittings to the PMMA stand.

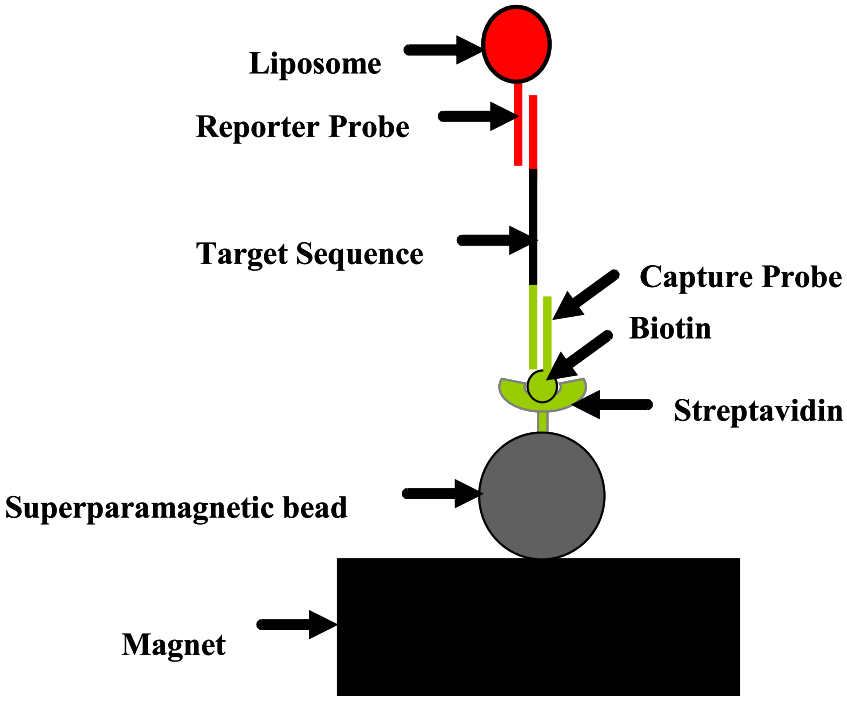


Figure 1.10: Principle of biosensor assay (not to scale). A DNA reporter probe is coupled to a liposome while a capture probe is coupled to a magnetic bead. When target RNA is present, the probes hybridize to it. The liposome-RNA-bead complex is subsequently immobilized on a permanent magnet in the capture zone of the device.

Two different DNA probes complementary to unique sequences on the target

pathogen's RNA serve as the biorecognition elements. For signal generation and amplification, one probe was coupled to dye encapsulated liposomes while the second probe was coupled to superparamagnetic beads for target immobilization. In the presence of target RNA, both probes hybridized to the target, forming a liposome-target-bead complex. This complex was subsequently immobilized in the magnetic field of the embedded magnet (Figures 1.9 to 1.11). The amount of liposomes captured correlates to the concentration of target sequence and was quantified using a digital camera mounted on a fluorescence microscope. Dengue fever virus serotype 3 sequences and probes were used as a model analyte system to test the sensor. Probe binding and target capture conditions were optimized for sensitivity.

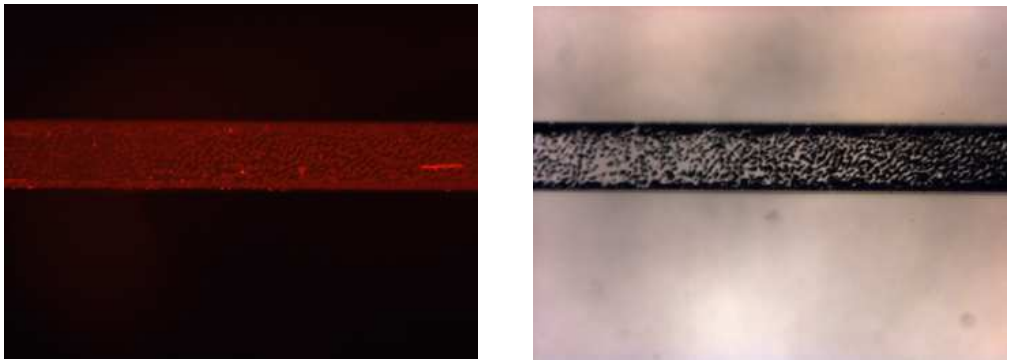


Figure 1.11: Fluorescent (a) and brightfield (b) images of captured liposome-RNA-bead complexes. The fluorescence is due to the sulforhodamine B dye encapsulated by the liposomes.

Improving on the biosensor design and assay, the second prototype achieved significantly better quantitative performance with pre-incubation of the test solution, liposome lysis and most importantly, control of flow rates by replacing the simple vacuum suction technique with a programmable syringe pump.[97] This microfluidic biosensor was then used for the rapid, sensitive, and serotype-specific detection of Dengue virus.[98]

The optical biosensor requires an expensive microscope equipped with a camera, fluorescence filters, other electronics and a computer to measure and quantify the fluorescent signal. While some attempt has been made to miniaturize the components of optical sensors, the majority of reported miniaturized systems using optical detection still rely on large fragile external components that preclude field operation. Equipment for an electrochemical sensor-transducer can be simpler, less expensive and portable.

1.5.6 Electrochemical transduction

Electrochemical transduction refers to the production of electrical energy by chemical reactions. Primarily, this is in the form of charge transfer between an electronic conductor (electrode) and an ionic conductor (electrolyte). Charge is transported through electrodes and electrolytes by the movement of electrons and ions respectively. Electrodes can be made out of carbon, metals such as gold and platinum or semiconductors such as silicon. While solid or non-aqueous electrolytes exist, the most widely used electrolytes are solutions of ionic species such as sodium (Na^+) and chlorine (Cl^-) in water.

A pair of electrodes in an electrolyte is referred to as an electrochemical cell. This is the most basic setup for an electrochemical transducer. One class of electrode reactions in an electrochemical cell is the oxidation and reduction of two chemical species in solution. The chemical pair is called a redox couple. Oxidation occurs when the potential of one electrode is sufficiently positive that one of the chemical species in the redox couple donates electrons to the electrode. Conversely, reduction occurs when the potential of an electrode is sufficiently negative that the second species in the redox couple accepts electrons from the electrode. The positive electrode is designated the anode, and the negative electrode, the

cathode.

An example of a redox couple is iron metal. In solution, iron metal can exists in a doubly or triply charged ionic form Fe^{2+} or Fe^{3+}. In the presence of an inert electrode substrate such as platinum or gold, Fe^{2+} will be oxidized to Fe^{3+} at the anode while Fe $^{3+}$will be reduced to Fe^{2+} at the cathode.

Anode: oxidation

$$Fe^{2+} \longrightarrow Fe^{3+} + e^{-} \tag{1.1}$$

Cathode: reduction

$$Fe^{3+} + e^{-} \longrightarrow Fe^{2+} \tag{1.2}$$

While this particular iron oxidation-reduction occurs naturally, it is an extremely slow endothermic process. If energy in the form of a potential between the electrodes is applied, the otherwise non-spontaneous electrode reactions proceed quite rapidly. Many metals, metal complexes as well as many physiological molecules and pharmacologically important chemicals exhibit similar reactions in response to specific applied potentials. Thus, it is possible to discriminate between different redox couples with an appropriate selection of the applied potential. This principle is commonly used in electrochemical measurements but with three electrodes instead of two. When a current of i is flowing between two electrodes due to electrode reactions, the potential drop of the cell itself is iR where R is the resistance of the solution. For this reason, it is difficult to precisely control the potential difference with a conventional 2-electrode system. In a 3 electrode system, the excitation potential difference is maintained between two electrodes, i.e. the working and reference electrodes. The current generated as a result of electrode reactions is measured between the working electrode and a third electrode, referred

as the counter electrode. The instrument that controls the potential between the working and reference electrodes and also measures the current flowing between the working and counter electrodes is called a potentiostat.

Electrodes can be classified into macro, micro and ultramicro categories depending on their size. Electrodes of area 1cm^2 or more are considered macroelectrodes while those in between 1 cm^2 and 1 mm^2 are characterized as microelectrodes. Electrodes with dimensions under 1 mm^2 are referred to as ultramicroelectrodes. Apart from size, microelectrodes differ from macroelectrotrodes in ways that allow their use in simpler setups with more sensitivity and selectivity. Macroelectrodes for instance have significant ohmic (iR) drop problems and reactant depletion within their diffusion layers. Microelectrode geometries on the other hand, can be designed to avoid these problems.[99]

With a potentiostat, different kinds of excitation signals can be applied to electrodes and the current response monitored to study the kinetics of reactions in solution. This branch of electrochemistry where current is measured under the application of a potential is referred to as amperometry. It include techniques that employ variable excitation potentials such as cyclic voltammetry, differential pulse voltammetry, chronoamperometry and constant potential amperometry (sometimes referred to as direct current potential amperometry, DCPA). Constant potential amperometry is the method used in most glucose monitors.

Current is not the only measurable quantity in electrochemical transduction. Measurements can be based on a change in the measured voltage between electrodes (potentiometry), or a change in the ability of a solution to conduct charge (conductometry) under controlled current conditions. The i-STAT analyzer previously discussed, employs all these methods of electrochemical transduction schemes with different cartridge configurations. It measures concentrations of sodium,

potassium, ionized calcium, carbon dioxide and pH with ion-selective electrode potentiometry. Hematocrit, or the volume of a blood sample occupied primarily by red blood cells (RBC), is determined conductometrically. Blood oxygen and glucose concentrations on the other hand are measured amperometrically.

1.5.7 Electrochemical detection of nucleic acids

Electrochemical nucleic acid sensors use the same hybridization detection principles as optical sensors with an electrochemical transducer. This involves the immobilization of an oligonucleotide probe onto or near an electrode, hybridization of or probes to a complementary target sequence, and transduction of the hybridization event. Electrochemical detection strategies fall into roughly three categories: direct oxidation/reduction of the sample nucleic acid, indirect detection with electrochemical mediators or indirect detection with the nucelic acid acting as a charge transport mediator.[100–102]

The bulk of direct detection methods reported to date, use no indicators but rely on the intrinsic electroactivity of the nucleic-acid base guanine. Wang et al, demonstrates such a direct detection, indicator-free method with an assay for *Cryptosporidium parvum* using inosine-substituted, guanine-free capture probes.[103,104] The probes were immobilized on a carbon paste electrode prior to potentiometric measurements at a constant current of 6 μA. Inosine, like guanine, binds preferentially to cytosine bases, but, has an oxidation peak well seperated from the guanine peak. Monitoring the potential at the working electrode for the formation of a guanine oxidation peak when the sample solution is added, gives good measure for the amount of target DNA in solution. This study achieved a detection limit of about 120 ng/ml of target DNA. While the indicator-free method is simpler, it is a direct hybridization assay and results in higher non-specific binding and thus

less sensitivity compared to other protocols.

Indirect methods with electrochemical mediators are far more popular than the other two strategies for nucleic acid detection. They often mimic fluorescent detection schemes by simply replacing the optical reporter with an electrochemical one. Hybridization is thus indicated by the characteristic response of the electrochemical reporter. The earliest electrochemical DNA sensors, such as the one reported by Millan and Mikkelsen in 1993, used this method.[105] Their sensor's transduction was based on tris(2,2-bipyridyl)cobalt(III), tris (1,10-phenanthroline)cobalt(III) perchlorate and Co(phen)3(3+). These electroactive reporters binds more strongly to double-stranded DNA (dsDNA) than to single-stranded DNA (ssDNA) in an intercalative mode, thus the voltammetric current signal increases if the sample solution contains the target sequence. Other electrochemical mediators employed have included redox-active molecules like ferrocene;[106–108] nanoparticles such as colloid gold;[109–111] and enzymes such as horseradish peroxidase that catalyze the formation of electroactive molecules or precipitate electrochemically inactive molecules to quench a signal.[112,113]

An interesting variation of electrochemical reporter transduction somewhat similar to the use of molecular beacons in optical detection is the use of a stem-looped capture probe that partially hybridizes to itself in the absence of a complementary target sequence.[114–116] In an assay developed by Fan et al., one end of the capture probe was labelled with a redox-active ferrocene molecule and the other end was immoblized on the surface a gold electrode. The looped structure allowed ferrocence to come into contact with the electrode and undergo redox reactions that produced a measurable current. In the presence of a target nucelic acid sequence, hybridization to the probe forced the ferrocene away from the surface of the electrode thus reducing the measurable current in inverse proportion to the

concentration of the target.[116]

The third approach to electrochemical detection takes advantage of the inherent ability of double stranded, but not single-stranded nucleic acids, to transfer charge from an electrode to a redox-active ligand (called an intercalator) that binds electrostatically to the strand. Hybridization of target sequence to a capture probe immobilized on an electrode creates a 'wire' that allows charge to flow to an attached intercalator. Steel and co-workers demonstrate the concept with a protocol that incorporates ruthenium hexamine in the reagent solution.[117] As DNA is captured by probes immoblized on gold electrodes, proportionately more ruthenium hexamine binds to the DNA, thus, generating a proportionately higher signal. A relatively more complicated but highly sensitive variation of the technique uses electrocatalytic reduction of ferricyanide by the intercalator methylene blue.[118] On hybridization, charge is transferred through perfectly matched pairs of DNA, to the intercalator methylene blue which is subsequently reduced. The reduced form of methylene blue then reduces ferricyanide thereby amplifying the electrochemical signal that could be produced with only the intercalator. In the absence of target sequence or hybridization with a mismatch, methylene blue is not catalytically active and the electrochemical signal is greatly diminished.

1.6 Form Of Solution

This dissertation discusses an application-led research effort to combine electrochemical biosensors and nucleic acid hybridization assays with microfluidics and microelectronic technologies towards developing a portable micro total analysis system suitable for pathogen detection in clinical, environmental and food safety applications.

The electrochemical detection scheme designed for the pathogen identification

and quantification application replaces the optical biosensor and microscope instrumentation with an ultramicroelectrode array (IDUA) and a miniaturized potentiostat optimized for constant potential amperometry. The microfluidic cassette was modified to accommodate the IDUA while the redox couple of potassium ferri and ferrohexacyanide ions replaced the dye encapsulated within the liposome reporter particles. As in the optical detection scheme, target RNA was immoblized on a magnet within the microfluidic device. The liposomes were then lysed after capture to release the ferri and ferrohexacyanide molecules unto the IDUA. Powered by the potential maintained by the potentiostat, the subsequent oxidation and reduction of ferri and ferrohexacyanide produced a measurable current proportional to the quantity of target RNA in solution.

Chapter 2 documents the requirements that guided the design of the system. Chapters 3, 4, and 5 discuss the design and implementation process. Chapter 3 emphasizes the hardware technical design; chapter 4 discusses test results with convenient laboratory samples while chapter 5 focuses on design of the system's embedded software. Chapter 6 discusses the design and implementation of a PC application that includes a graphical user interface for interaction with the device. Chapter 7 details the modifications made to the basic detection system to allow multiple sensor detections. Chapters 3, 4, and 7 were written for publication as journal articles so there may be some redundancy in the information presented. Chapter 8, the concluding chapter, summarizes the milestones achieved and discusses technologies that can be integrated to improve the system.

1.7 Explicit Limitations

The device prototype encapsulates target capture, detection and data analysis/display. It does not integrate the sample preparation and target amplification pro-

cesses. Thus while it is immediately useful as a detection system, the user is required to have a concentrated sample or manually perform the sample preparation and amplification steps. The system was however designed to be a modular and flexible platform that can be easily upgraded and modified as needed.

1.8 Contributions

The primary contribution of this dissertation is the construction of a miniaturized electrochemical detection platform by combining liposome signal amplification for electrochemical detection of a target molecule on a microfluidic chip with a microelectronic circuit for control, signal processing and data storage. The resulting prototype addresses the problem of detecting pathogens via their RNA with an accurate inexpensive handheld device. It has significant advantages of speed and simplicity over the laboratory-based optical system and could be of considerable use for inexpensive point-of-care diagnostic assays, rapid in-plant food safety testing, genetic screening and field testing for environmental contaminants and biowarfare agents. The prototype was designed from the start for production as a mass market device, so, commercial concerns such as costs, energy efficiency, and ease of use were considered as part of the system design at an integral level. The modular architecture employed to develop both the hardware and software will help other researchers use the system to quickly and easily implement solutions to other portable detection applications.

CHAPTER 2

APPLICATION REQUIREMENTS

The goal of this research effort is to develop a mass market electronic appliance featuring a miniaturized electrochemical transduction system (miniEC) with integrated recording and data manipulation to replace the lab-based fluorescence detection of RNA set-up with a microscope described in the introductory chapter. While conventional electrochemical set-ups can be as bulky as fluorescence detection systems, the former has the potential for cost effective miniaturization that is as yet unmatched by the fluorescence detection. This potential for portability and simplicity is demonstrated by the phenomenal success of personal glucose monitors in the marketplace. The subsequently described requirements and design for the system are tailored for a microbiosensor application for the detection of RNA isolated from pathogenic organisms. However, the envisioned final device, colloquially referred to as the miniEC, is far more general and can be used for multiple applications of electrochemical detection in fields as diverse as clinical diagnosis, food safety testing, and environmental monitoring.

Like the glucose monitors, the miniEC should be portable, easy to use and battery operated even as it replaces electrodes, potentiostat and computer systems in conventional laboratory electrochemical set-ups. One reason often cited for the success of glucose monitors on the market is the low cost of production. Thus designing the miniEC for low cost and mass production are equally important objectives. Also, as a mass market device, the miniEC is intended for portable electrochemical measurements by an average user, as such, it should not require any special skills beyond those required to mix reactants or any understanding of the underlying design technologies. Furthermore, the device will be integrated with other active modules of a micro-total analysis system such as a micro-pump

actuator and a heating block for nucleic acid amplification. Thus, it is important for it to be structurally and functionally modular so it can easily be maintained, modified or upgraded for other electrochemical applications.

2.1 Design Concept

The miniEC design will be achieved with a combination of customized microfabricated components and commercial off-the-shelf microelectronics assembled for RNA detection, signal measurement and control. The concept that best describes the architecture of the miniEC is illustrated in Figure 2.1 below. It is made up of five subsystems:

- The biosensor unit
- The control unit
- The controlled unit
- The subsidiary units
- The user interface

The biosensor unit is a microfluidic cartridge similar to the one described in the introduction for the fluorescence detection setup. For the miniEC system, it will be modified to include an array of ultramicroelectrodes that transduces the amount of target molecules present into an electronic signal that the control unit can read. The principle of detection is the same as it is for the fluorescence system. Target molecules are bound to liposomes filled with the redox couple potassium ferri and ferrohexacyanide. When the liposomes are lysed, the released electrochemical molecules undergo redox reactions on the electrode array, producing a measurable current signal within 1 minute. The electrode array is powered by a potentiostat

that maintains an excitation potential across the electrodes to drive the redox reactions. The potentiostat also measures the produced current.

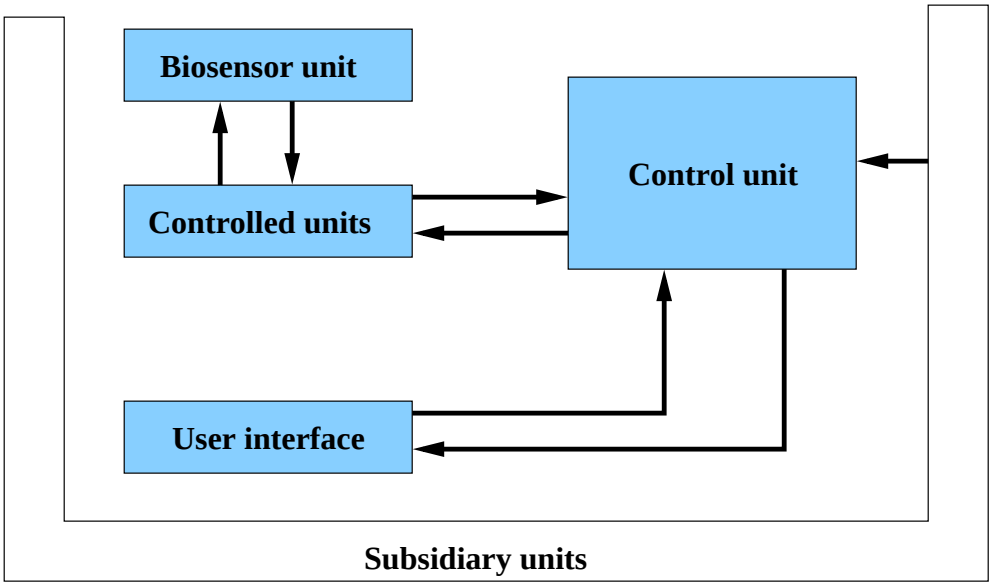


Figure 2.1: Architecture of the miniEC showing the direction of data flow between the biosensor unit, control unit, user interface and controlled units whose functions define the miniEC. The subsidiary units (for example power supply) support the functioning of the other units.

The control unit is the heart of the system. It receives and processes signals from the biosensor and controlled units and in return, sends back commands or signals that determine the next step in operation of the system. It also interprets commands from a user through the user interface and returns the appropriate status messages after performing or directing the controlled units to perform the functions requested. The controlled units refer to other sensors and actuators apart from the biosensor such as the potentiostat, (and in the future, other components such as a micropump) that are supervised by the control unit to achieve the miniEC's core functions. The subsidiary units refer to auxiliary systems of the device, such as the power system and the controller's programming interface that perform necessary but not defining miniEC tasks. The user interface provides the means for a user to communicate with the system.

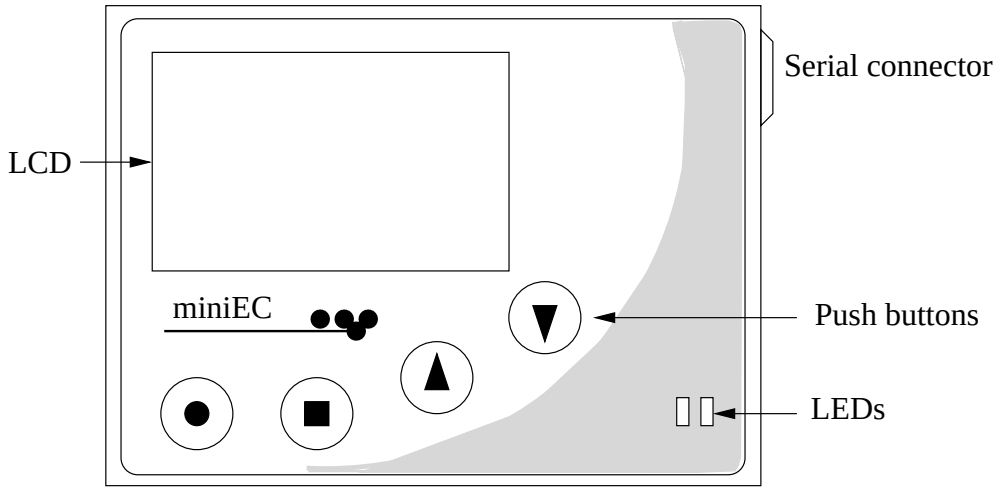


Figure 2.2: Conceptualized external appearance of the miniEC featuring an LCD screen to display data and operational status, buttons for user control, a slot or connector for the biosensor and a serial connector for interaction with a PC.

Figure 2.2 shows a possible prototype of the miniEC. The most visible elements are those of the user interface which includes push-buttons, a full function LCD screen and a serial communication connector for PC networking. The miniEC will also have a slot or connector for the biosensor cartridge. The rest of the device is enclosed and hidden from the user. The buttons will be used to start or stop experiments, or in conjunction with the LCD menu, to configure the system. Recorded data can be read directly off the LCD screen or be transmitted to a host PC for further review and long-term data storage.

Design and implementation of the miniEC application uses a systems-based approach which incorporates a high level of modularity in the hardware and software partitions. The specific design of the system is beyond the scope of the chapter and is covered in the three following chapters. The rest of this chapter provides a comprehensive specification of the requirements guiding the various design and implementation decisions. These specifications include functional requirements for system behavior; usability requirements, which identify the type of interfaces available to users and the accessible functionality of such interfaces; hardware and

firmware requirements for the components used to build the system; and performance requirements which specify minimum required throughput, capacity and resource utilization.

2.2 Functional Requirements

Constant current potential amperometry (or direct current potential amperometry, DCPA) is the simplest of electrochemical detection tests that can be performed with a redox couple. In DCPA, a constant potential is maintained across the electrochemical cell, and the current response is monitored. With an excitation potential of 400 mV driving the redox couple of potassium ferri and ferrohexacyanide, a steady state current dependent on the concentration of ferri and ferrohexacyanide in a non-stirred solution can be recorded within one minute.[119] Using this baseline, the miniEC at its most fundamental level should allow a user to take and view 1-second interval measurements of current from the transducer for a duration of 60 seconds at an excitation potential of 400 mV. Taking data at 1-second intervals allows the user to see the course of the signal as it approaches steady state. Test parameters of 400 mV excitation potential, 1-second interval signal readings for 60 seconds duration is referred to as a basic test in the rest of this document.

To extract the functional requirements, the desired behavior of the system is considered from the user's point of view. The use case diagram in figure 2.3 summarizes the interaction between a user and the miniEC system. In addition to the obvious task of allowing a user to run a detection experiment, the miniEC is required to run processes that allow a user to view and set time, view and modify experimental parameters, and to view, retrieve or delete recorded experimental data. The functionality of the miniEC extrapolated from the use cases fall into categories relating to the user interface, configuration, experimental and data

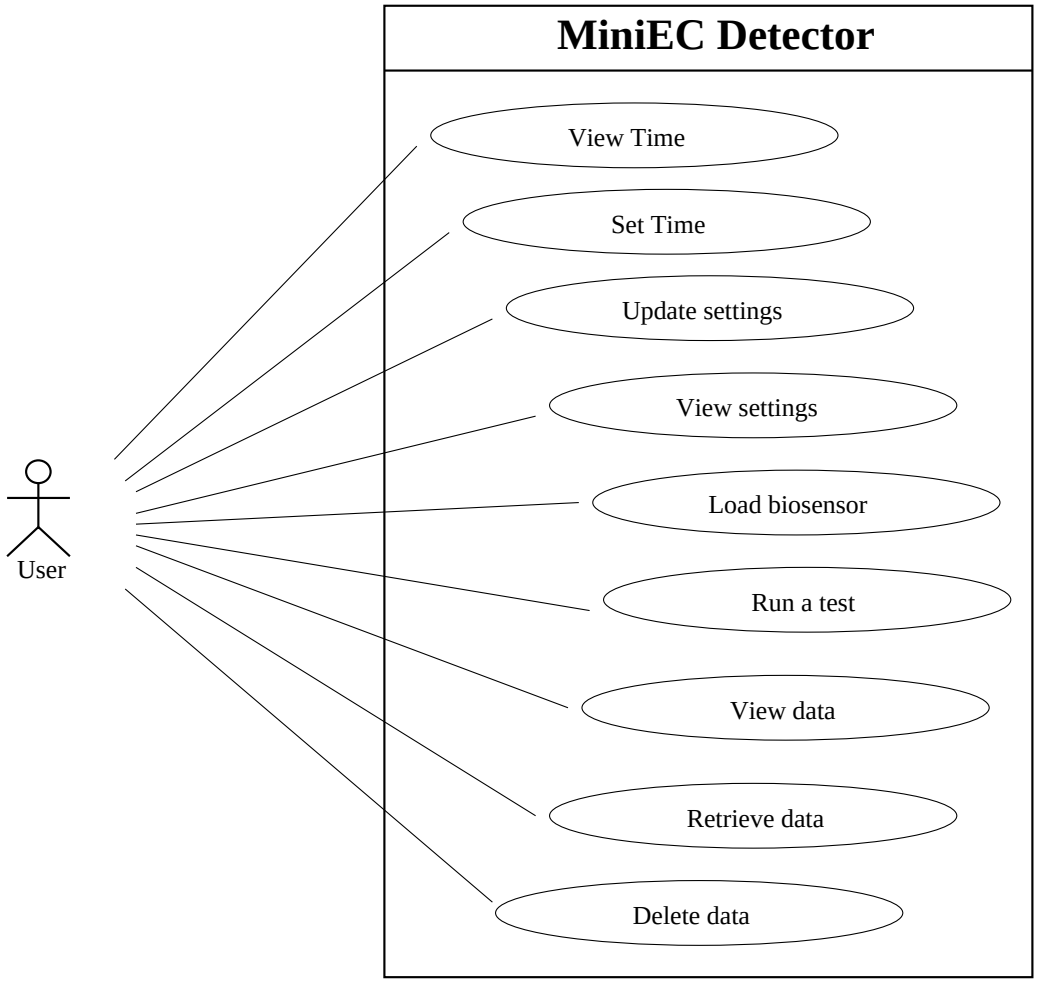


Figure 2.3: Use case diagram of the miniEC detection system showing the services provided by the miniEC system to a user. The user may view or set the time and experimental test parameters, run a test and view, retrieve or delete stored test data.

management services.

2.2.1 User interface

Display

The miniEC must incorporate an LCD display to prompt the user, provide a menu for operation, and allow the user to easily view device status. A graphical user

interface on a PC is also highly desired for interaction when it is networked to a miniEC.

Input mechanism

The user requires a clearly labeled push button or keypad interface to interact with miniEC hardware and firmware. At a minimum four buttons are required: a run button to start an experiment, a stop/enter button to stop an experiment or confirm a menu selection, a button for selection of an item from a list of menu options and a reset button to return the miniEC to default settings.

2.2.2 Configuration

View and set time

The miniEC must support an internal clock to keep time to the year, month, day, hour, minute and second. Keeping track of the passage of time is important because timer interrupts initiate activity in the system and stored data files must be labeled with their timestamps. The miniEC must allow a user to set and adjust the time to the correct time zone at any time except when an experiment is running. The miniEC should also be able to update its clock through synchronization with a PC through a graphical user interface.

View and set test parameters

Reasonable default settings for time and test parameters must be included in the miniEC firmware depending on the intended end use. The current test parameters must be displayed when requested by the user. The user can adjust these test parameters from a menu on the miniEC or via the graphical user interface on a

connected PC. Viewing and changing test parameters cannot be performed when an experiment is running.

2.2.3 Running experiments and data management

Load a sensor module

The electrochemical biosensor module of the miniEC system should be located on a separate cartridge apart from the rest of the system's electronics yet maintain secure electronic contact for measurements. It should be connected in such a way that the user can easily load samples and reagents for the electrochemical tests using an external pump. Separating the cartridge in this manner will prevent cross contamination between experiments as the cartridge can be made of disposable materials for single use experiments or easily removed for cleaning between tests.

Creating and storing files

The miniEC must create and store files of tests run by a user. If the miniEC is run attached to a PC, the data need not be stored on the miniEC but could be transmitted directly to the PC.

Monitoring data

The miniEC must display results in real time on the LCD. If connected to a PC, the miniEC must display results in near real-time on the PC as well.

Viewing file information

The user may view a summary of test data on the LCD. For a look at all the data in a file, the user can use the graphical user interface to retrieve and view stored

data. The test data files can also be deleted by the user at any time via either the GUI or the hardware user interface.

Networking

The miniEC must be able to communicate with a PC via serial RS-232, USB or both.

2.2.4 System interaction and integration

The system should include sufficient resources to accommodate future functional upgrades and integration with other components of a micro-total analysis system. The potential upgrades may include control of one or more micro-pumps for controlled fluid flow, heating blocks for nucleic acid amplification, or hardware for wireless communication.

2.3 Performance Requirements

2.3.1 Size

The miniEC must be compact and have a physical profile suitable for hand-held use, as such, form and weight are important characteristics. A palm-sized personal digital assistant (PDA) or pocket PC makes an ideal size model for the miniEC. The average PDA has width and length dimensions of 75mm x 100mm (3 x 4 inches), weighs less than 8 ounces and is under 20mm thick. These dimensions are adopted as the upper limit of the miniEC size.

2.3.2 Power

The miniEC requires power autonomy as a portable device. This requires the use of batteries as its power source. A long battery life greatly enhances the usability of a handheld device since it has a direct impact on the duration of mobility for the device user. Larger, denser batteries offer longer use between changes, however, restrictions on the miniEC size and weight limit the size and density of batteries that can be included. While this specification does not insist on the use of a particular battery, the battery selected must be readily available in retail stores. The average glucose monitor battery allows the user to perform about 1000 tests before it needs changing. This specification is adapted as a target for the miniEC. In short, the miniEC requires the use of a readily available, small, low weight battery pack capable of supplying enough power for 1000 basic tests. Furthermore, the miniEC should draw less than 100 mA at 3V supply even under maximum usage conditions. This will make it possible for it to draw power from the PC when it is connected via USB, thus prolonging battery life.

2.3.3 Initialization

The miniEC should initialize or reset to a known hardware and software state when power is applied to maintain integrity of results after power interruptions.

2.3.4 Graceful failure

The device must have a reset function in hardware and software to reset it to the default settings when the user detects problems with the current settings.

2.3.5 Operating environment

While the miniEC must be configurable and operable via a PC, the PC is not required for proper and full use of the device. The device must be able to function properly as a stand-alone device in a home, lab or office environment in point-of-care, clinical, food testing and environmental monitoring applications. The ambient conditions in these settings require that the miniEC must operate properly within the standard temperature and non-condensing humidity range of 0 to 70 C and 20 to 80% respectively for commercial products. These specified ranges may be exceeded if it does not cost significantly more to do so.

2.3.6 Memory

Since the device may be used in many different ways, it is difficult to stipulate data storage in terms of time (for example, a week's worth of data) as is done for glucose monitors. Instead, we stipulate that the device should include enough non-volatile memory to store data from an arbitrary target of 100 basic tests.

2.3.7 Electrochemical effectiveness measures

The BAS EC epsilon electrochemical workstation is a popular, highly respected standard piece of equipment in the electrochemical industry. While the miniEC will have a fraction of the BAS capability, its performance in DCPA detections must be verified with identical experiments performed with the electrochemical workstation.

The range of current measurement required is from 1 nA to 1 μA. Previous experiments show that range of current readings produced by the typical IDUA in our lab is between 1nA to 800nA for about 10 μL of potassium ferri/ferrohexa-

cyanide concentrations of 0 to 100 μM.[119] This volume of redox couple is 3 orders of magnitude greater than the volume estimated to be required by the microfluidic setup. Furthermore, in the dynamic environment of the microfluidic cartridge, the concentration of the redox couple will be diluted by running buffer solutions. Thus the current measured is unlikely to exceed 1000 nA. Over time however, the miniEC will be enhanced, therefore the design should make provision for ranges that extend beyond 0 – 1000 nA.

The range of potential excitation required is from 0 to 1.25 V in 10 mV steps. This includes the 400 mV used with the ferri/ferrohexacyanide redox couple and a wide potential range for many other redox couples.

The range of intervals in seconds that can be selected for tests must include the intervals 1, 2, 5, 10, 30 and 60. These are the intervals supported by the epsilon workstation that are typically used in electrochemical experiments.

2.3.8 Cost

The final cost of the electronic components should not exceed $50 for production quantities.

2.3.9 Accessories

The user should not need extra hardware or software apart from that packaged with the miniEC system.

2.4 Hardware and Firmware Requirements

Hardware refers to the actual physical electronic components that make up the miniEC while firmware refers to the software programmed into the hardware.

2.4.1 Microcontroller based hardware design

To keep development and repair costs low, commercial off-the-shelf components should be used instead of developing customized electronic components. Integrated solutions are also preferred over discrete components since every component in the system takes up space, uses power and generates heat. Component count therefore drives production costs and ultimately the final cost of the device. Subsequently, a major objective is to reduce the component count to a minimum. Incidentally, a reduced component keeps the system compact, ensuring a design with a physical profile suitable for hand-held use. A general purpose microcontroller-based design is thus ideal.

A microcontroller is essentially a miniaturized computer that packs a great deal of functionality into a small space. It can provide program and data memory, op-amps for signal conditioning, analog-to-digital conversions, clocking capabilities and general input/output functions among others. An added benefit is that a lot of the system's functionality can be built in embedded code thus eliminating problems associated with hardware overhead, integration and upgrading.

Furthermore, selection of a microcontroller with on-chip re-programmable flash memory ensures flexibility in the design, production and end use of the miniEC. Consequently, differentiated products can easily be generated for applications in niche fields within a short time-to-market window.

2.4.2 Hardware standards

The miniEC board must conform to IEEE 1149 standard (JTAG) in-system flash programming and debugging. JTAG specifies debug circuitry typically built into CPUs that enables inexpensive programming and testing of circuit boards through a small number of dedicated test pins. It also provides a way for upgrading the

miniEC's embedded code at any point in time after the initial deployment.

Hardware must be compatible with RS232 and USB communication standards for networking so that generic communication cables and protocols can be used.

2.4.3 System interaction and integration

The system should include sufficient resources such as I/O pins left over or bus protocols such as SPI and I2C to accommodate future functional upgrades and integration with other components of a micro-total analysis system. The potential upgrades may include control of one or more micro-pumps for controlled fluid flow or heating blocks for nucleic acid amplification.

2.4.4 Firmware programming language

To facilitate debugging and future migration of the firmware to a more powerful microcontroller, the firmware must be written in a high level human-readable standard programming language like ANSI C. This requirement will also enable development on any modern operating system platform.

2.4.5 Programming style

The firmware must be modular in terms of tasks the MCU must perform and fully documented for easy code debugging, maintenance and upgrading.

2.4.6 Timing

The miniEC is primarily a reactive system, in that it responds to actions initiated by the user. As such, the firmware must assert interrupts in near real-time when

required or requested by the user interaction. For example, pressing a button to run an experiment should start the experiment within a second.

2.5 Operator Interaction

There are two parts to the user interface of the miniEC. There is the hardware interface of the LCD coupled with buttons that allow the user to confirm status and interact directly with the miniEC firmware and a graphical user interface (GUI) that runs on the user's PC. While not entirely necessary, the provision of a GUI makes interaction with the miniEC easy for those familiar with a PC.

2.5.1 Physical interface

Display

The miniEC must incorporate an LCD display to visually confirm device operation, display time of day when not in operation and display results of running tests. The LCD will also display a menu for operation, provide status messages and prompts to help the user operate the device and allow the user to easily view device status at any time.

Input mechanism

The user requires a clearly labeled push button or keypad interface to interact with miniEC hardware and firmware. Each button will generate a unique event that will provide the user with the ability to perform specific actions such as set initial time and date, set interval, potential and duration test variables, start/stop a detection experiment, or view archived data.

2.5.2 Graphical user interface

The miniEC GUI resides on the desktop computer connected to the miniEC over a serial RS232 or USB connector. It provides an alternative control and display interface in the familiar environment of a PC desktop.

Control interface

The GUI's control interface is responsible for establishing a connection to the miniEC hardware, downloading data from the miniEC memory, starting tests, stopping tests and updating parameters such as time, excitation potential, etc. on the miniEC. It also should allow the user to query the miniEC hardware to identify the model and the version of firmware running on it.

Display interface

The display interface is responsible for reading data sampled by the miniEC and performing any analysis or transformations required to display and chart the results over time. It will also display charts of previously recorded data and allow the user to export the data in the PNG image format. In addition, it will allow the user to export data as plain text or in XML format for use in third party analysis tools.

Operating system

Cros- platform executable installation and operation on Windows and Linux operating platforms for the GUI is required. Operation on the Macintosh platform is also desired.

Programming language

To keep costs low yet enable rapid development, the programming language selected needs to have royalty-free libraries for generating code for graphs, portable network graphics files, and serial input-output communications to hardware.

2.6 Development Strategy

The requirements presented here were not provided by a client as defined in the traditional sense of systems engineering. They instead evolved out of our laboratory's need for a miniaturized electrochemical detection system as the next component in a micro-total analysis system and the need in general, for inexpensive field portable electrochemical equipment. As mentioned several times, the successful electrochemically based glucose monitoring system was chosen as a comparable product available on the market for some of the design criteria in addition to the criteria developed in the lab. As such, the development approach to achieving the application requirements documented above was by necessity iterative. It involved a sequence of designing, building and testing until the prototype was complete.

CHAPTER 3

AN EMBEDDED SYSTEM FOR PORTABLE ELECTROCHEMICAL DETECTION*

3.1 Introduction

Many successful miniaturized analysis systems for industrial and biomedical applications have been reported in literature over the last decade.[120–124] Apart from glucose sensors however, mass market production of miniaturized or "lab-on-chip" systems for portable applications has not proceeded as rapidly as expected. The first of such systems are just beginning to appear. Examples include the i-STAT portable analyzer[30] and Sandia's MicroChemLab biotoxin detector.[29] The i-STAT performs a wide selection of blood tests using disposable microfludic cartridges. MicroChemLab is in field use in the Boston subway transit system (Massachusettes, USA) and used for monitoring water systems in other parts of the United States. Another example notable in particular for its low cost is the $40 Pria Diagnostics male fertility kit.[125]

The majority of portable systems reported use electrochemical detection.[29,120–124] The sensitivity and selectivity of electrochemical detection are comparable to spectroscopic detection methods.[100] In addition, electrochemical detection, particularly amperometry, offers key advantages in simplicity, smaller size and lower cost. In amperometry, a potential is applied by a potentiostat to an electrochemical cell to cause oxidation-reduction (redox) reactions of electrochemically active molecules at the electrodes. The redox activity generates a measurable current that is proportional to the concentration of the electrochemically active molecules. The sim-

*AN ABBREVIATED VERSION OF THIS CHAPTER WAS PUBLISHED AS "AN EMBEDDED SYSTEM FOR PORTABLE ELECTROCHEMICAL DETECTION" IN THE JOURNAL SENSORS AND ACTUATORS B: CHEMICAL, VOLUME 123, ISSUE 1, 10 APRIL 2007, PAGES 336-343

plicity of the setup is demonstrated by the wide range of sub $100 amperometric glucose sensors available. Incidentally, the glucose sensor is the most abundant miniaturized test system in use today. It's success is attributed to the fact that the sensors are small, inexpensive, stable, easy to use, and easy to produce.[126]

Conventional potentiostats are designed for research and are capable of performing many different kinds of complex electrochemical analysis. As such they typically are bulky and therefore unsuitable for integration with the microfluidic technology used to miniaturize liquid assays. Several groups have designed and fabricated miniaturized single chip potentiostats from silicon that can be integrated.[127–129] The advantage of this approach is that potentiostats can be made very small, on the order of millimeters, and customized at fabrication specifically for specific experimental requirements such as current magnitude and frequency. For specialized applications such as in-vivo neurotransmitter monitoring, these integrated potentiostats may be the best option.[130] The setup and fabrication costs however, make this approach too expensive for the production of mass market analysis devices. The use of existing integrated circuits (IC) such as op-amps, microcontrollers and system-on-a-chip components is the least expensive way to achieve instrumentation miniaturization at minimum cost.[131,132] These components have already been optimized for microelectronic applications and are manufactured in so many different configurations that, with some research, suitable ones can be found to meet experimental requirements. When combined with micro-fabricated electrodes, devices with accuracy and sensitivity comparable to conventional bench-top electrochemical analyzers can be realized.[132,133]

The primary requirements of most portable battery-operated devices are low power consumption, low cost and small size.[134] Taking into consideration these and other business-driven factors such as long life-cycle and simple production, we

have designed a robust customizable microcontroller-based miniaturized system (miniEC) for electrochemical detection. Like other biosensing systems such as the glucose sensor, the miniEC has two parts. It consists of a disposable microfluidic cartridge that houses an amperometric sensor and an embedded system to power the sensor, as well as, measure, display and store the sensor results. This paper focuses on the design of the embedded system and discusses the rational for various design choices.

3.2 Hardware Design

3.2.1 Sensor

The basic requirement for an amperometric sensor is a pair of electrodes in an electrolytic solution. Maintaining a constant potential between electrodes while simultaneously measuring the amount of current passing through one electrode (referred to as the working electrode), provides information concerning the oxidation or reduction of electrochemical species at the surface of the electrode. Oxidation occurs when the potential of the working electrode is sufficiently positive that a chemical species in solution at the surface of the electrode donates electrons to the electrode. In contrast, reduction occurs when the potential of the working electrode is sufficiently negative that a chemical species in solution accepts electrons from the electrode. Both processes produce a current that varies with the concentration and identity of ionic species in solution and the applied potential maintained.

Advances in microelectronic technology that has made it possible to miniaturize potentiostat circuits, has also made it possible to greatly miniaturize the electrodes used in electrochemical detection. Microelectrodes have several advantages over

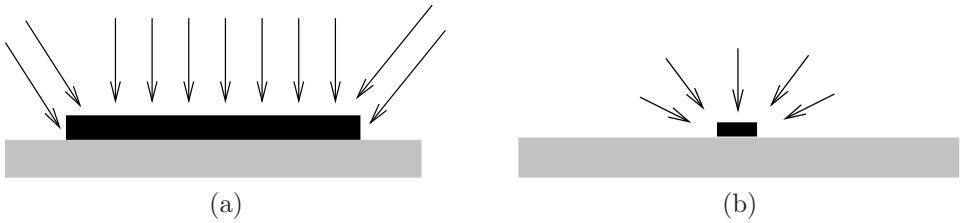
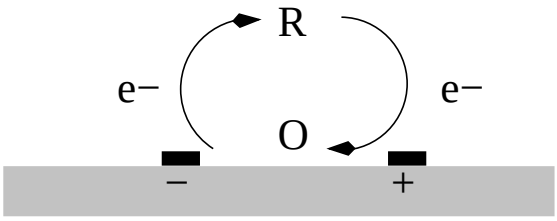


Figure 3.1: Diffusion layer profiles of (a) conventional macro- and (b) microelectrodes.

macroelectrodes due to the reduced size. Their small size for instance, allows them to be used with very small sample volumes, to be embedded into microfluidic applications as in this project, or used in *in vivo* experiments. Another advantage is efficient mass transport in the small volume and hence the rapid establishment of a steady state signal.[119] As an electrochemical species in solution is oxidized or reduced, a concentration gradient for the species is generated at the surface of the electrode. This region of the concentration curve is the diffusion layer. Figure 3.1 depicts a macro and a microelectrode and their associated diffusion layers at a constant potential. Arrows indicate new material brought to the electrode surface by diffusion from the bulk solution. Due to the small dimensions, the diffusion layer is radial over the surface and peripherial regions of a microelectrode. This property is particularly useful when the dimensions and spacing of a pair of microelectrodes is such that the diffusion layers overlap (Figure 3.2). In such cases, ions produced at the cathode diffuse to the anode where they are converted back to original form resulting in a sustained redox cycling process that produces a larger current signal. Sandison *et al.* systematically studied diffusion layers of adjacent microelectrodes and concluded that diffusion layers overlap when the inter-electrode distance does not exceed 10 times the electrode diameter.[135] The overlap of diffusion layers at these dimensions occur on a time scales of microseconds.[136] These dimensions can easily be created with modern microfabrication methods.

(a) Transport region



(b) Redox cycling

Figure 3.2: Illustration of (a) overlapping diffusion regions and (b) cycling of a redox couple, R and O between a pair of microelectrodes.

Compared to conventional electrodes, the current flow through a microelectrode is quite small even with redox cycling. They can however be fabricated in array configurations that retain the advantages of single microelectrotrodes pairs but as a unit, produce signals comparable to conventional electrodes. The amperometric sensor developed for this project takes advantage of this property. It consists of an interdigitated ultramicroelectrode array (IDUA) transducer embedded in a microfluidic channel (see chapter 4 for microchannel details).

The IDUA geometry pairs two combs of ultramicroelectrode so that they mesh with each other (figure 3.3). As such, it contains many repetitions of ultramicroelectrode electrode pairs spaced micrometers apart so that the transport regions of adjacent electrodes overlap. The IDUA sensor thus retains the advantages of microelectrodes, such as, the rapid establishment of a steady state signal; a large signal-to-noise ratio due to the small dimensions and diffusion pattern; and most importantly, very low ohmic drops between electrode pairs. Consequently, it can be used in a 2-electrode amperometric setup that simplifies the rest of the hardware design.

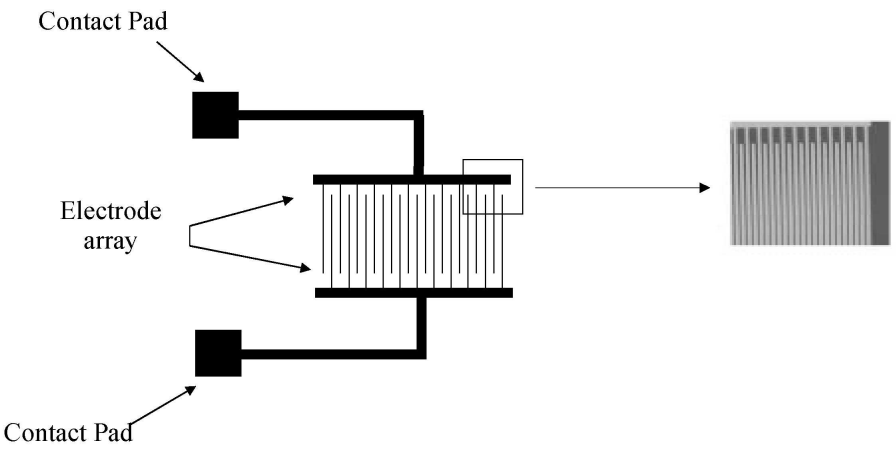


Figure 3.3: Schematic and optical micrograph of an interdigitated ultra microelectrode array (IDUA).

Figure 3.4 illustrates redox cycling in an IDUA. Current produced by the redox activity in each set of electrodes is additive over the array. Each electrode except for the two electrodes at the ends of the IDUA geometry are proximate to two electrodes of the opposite polarity. Ions produced at a cathode can diffuse to two anodes where they are converted back to original form. The redox cycling activity is thus more efficient in IDUAs.

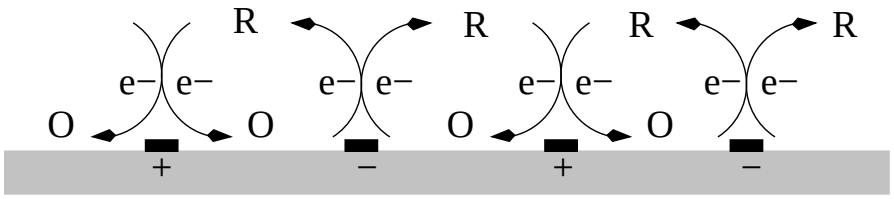


Figure 3.4: Illustration of redox cycling on an IDUA.

The IDUA is fabricated by standard clean-room photolithographic and lift-off techniques on glass wafers.[119,137,138] Briefly, a mask is prepared using a Pattern Generator that is five times the final size of an IDUA. Using a g-line stepper (436 nm) the pattern is transferred to Pyrex glass wafers (Corning 7740, Corning, NY) coated with positive photoresist (Shipley S1813, Shipley Company, Marlborough,

MA). Subsequently a 10 nm titanium adhesion layer and a 150 nm gold layer are deposited using an e-gun source (CVC 4500 Evaporator). The wafers are then soaked in acetone to lift off the excess metal and photoresist leaving the patterned electrode arrays. Before use, the glass wafers are rinsed with isopropanol followed by deionized water, dried and then diced into single IDUA chips. A variety of IDUAs were fabricated as described. Each had 400 to 500 pairs of fingers with a length of 1000 μm, widths ranging from 2.5 to 5 μm with inter-electrode gaps of 2.5 to 5 μm wide. These ranges were determined experimentally to be optimal for the indicator redox couple, potassium ferri- and ferrohexacyanide, chosen for fast reaction kinetics, at a potential of 400 mV.[119,138]

3.2.2 Microcontroller unit

The most important hardware component in the miniEC system is the the microcontroller unit (MCU). The microcontroller unit is essentially a miniaturized computer that is programmed to execute the behavior that defines the miniEC. An MCU was chosen that would meet miniEC system requirements for computational speed, low power consumption, development tools, design flexibility and low cost. Four MCU family architectures, AVR, PICmicro, MSP430 and MC68HC11/12 were evaluated by comparison of the Atmega128L (ATMEL, San Jose, California), PICLF452 (Microchip Technology Inc. Chandler, Arizona), MSP430F449 (Texas Instruments, Dallas, Texas) and MC9S12C64 (Freescale Semiconductor, Austin, Texas) microcontrollers considered at the time to be representative of the features offered by each family. The MSP430F449 was selected because it had the best combination of low power consumption, low leakage current, event-driven capability and selection of integrated low power peripherals for portable battery powered operation. Table 3.1 shows the most important subset of the microcontroller features

compared.

The MSP430 and 68HC11/12 have a 16-bit core CPU architecture compared to the 8-bit PICs and AVRs. This bit number is the length of data the MCU can handle with one instruction. An 8-bit microcontroller for instance can only process numbers from 0 to 255 thus multiple instructions are needed to work with larger numbers. The 16-bit architecture enables single transfer of 16-bit ($2^{16} = 65536$) values from peripherals or memory to registers and vice versa. Not only is this much faster but it is particularly useful in allowing variables to be updated within interrupt routines.

To keep all parts of a microcontroller system in sync, the CPU performs all operations in a timed sequence determined by a clock. Since digital transitions occur either at the rising or falling edge of a clock signal, the higher the clock frequency supported by an MCU, the more operations it can perform in a given time. The clock speed comparision in table 1.1 is a little deceptive. While, the PIC18 can be clocked at speeds up to 40MHz, it typically needs 4 clock cycles to complete 1 instruction. A PIC running at 20 MHz for instance, only performs 5 million instructions per second (5 MIPS). The 68HC11/12 typically needs 2 clock cycles per instruction so its maximum throughput at 16 MHz is 8 MIPS. The MSP and the AVR on the other hand can perform 8 million instructions at clock speeds of 8 MHz. For all these microcontrollers, the faster they are clocked, the more power they consume. So even though all 4 are capable of 8 MIPS performance, the PIC18 and 68HC11/12 will consume more power than the AVR and MSP430 to sustain that throughput. The MSP's ability to support 16-bit word data-memory access and ALU operations however, gives it an edge in data processing efficiency and over the AVR. Furthermore, the MSP430 microcontroller can generate their own clocks. For the greatest accuracy in synchronization however, at least one

Table 3.1: Microcontroller unit comparison.

Feature	AVR Atmega128	MSP430 MSP430F449	PICmicro PIC18LF452	MC68HC12 MC9S12C64
Core architecture	8-bit	16-bit	8-bit	16-bit
Power supply (V)	2.7-5.5	1.8-3.6	2-5.5	3.3-5
Power down modes	6	5	1	4
Clock (MHz)	0 - 8	0 - 8	0 - 40	0 - 16
Active current (mA at 1 MHz)	8	0.30	1.6	2
Idle current (μA at 32 KHz)	22	1	25	700
Flash memory (bytes)	128k	60k	32k	64
EEPROM (bytes)	4k	256	256	NA
RAM (bytes)	4K	2K	1.5K	4K
I/O pins	53	48	34	60
Pin leakage current (nA)	1000	50	500	100
Interrupts	35	40	18	29
JTAG interface	yes	yes	no	yes
Boot loader	yes	yes	no	yes (BDM)
Development tools (C-compilers)	avr-gcc (free)	mspgcc (free)	commercial ($$)	gcc-68hc1x (free)
Peripherals	Two 8-bit and two 16-bit timers; 10-bit ADC; two USARTs, I2C and SPI	Two 8-bit and two 16-bit timers; 12-bit ADC; two USARTs, I2C and SPI; 16x16 multiplier; LCD driver; Supply monitor	Two 8-bit and two 16-bit timers; 10-bit ADC; one USART, MSSP and PSP; 8x8 multiplier	One 16-bit timer and one 8-bit PWM; 10-bit ADC; SPI, CAN and SCI,

external crystal oscillator is needed to generate a clock signal. The flexible clock system of the MSP430 microcontroller enables a high clock frequency active mode (for fast execution) and 5 low power, lower frequency operating modes that are key to its phenomenal low-power operation. Figure 3.5 shows the dramatic power savings that can be achieved with each power mode from active mode (AM), to low power mode 4 (LPM4).

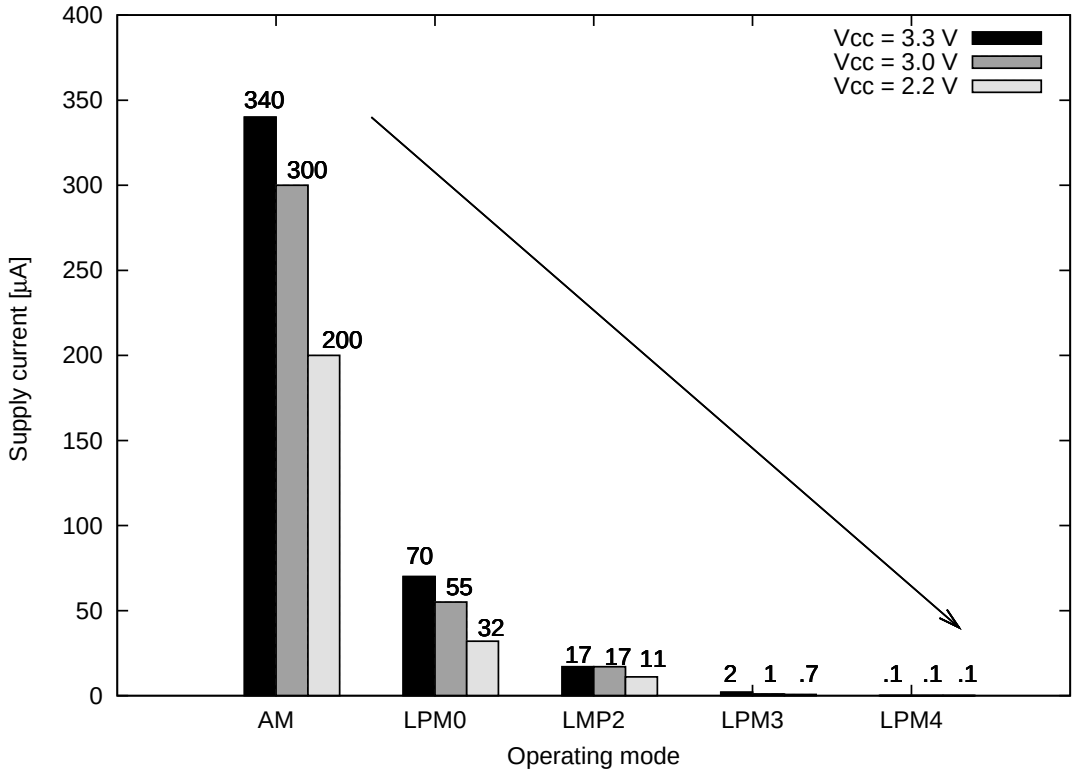


Figure 3.5: Typical current consumption of MSP430 microcontrollers in 5 operating modes from active mode (AM) to low power mode 4 (LPM4)

In the envisioned application, the MCU spends most of its time in idle mode with periodic bursts of activity when signals are measured, thus, power consumption in the idle mode and pin leakage current are the most important features that determines if one operates within a low power budget. The MSP430 in idle mode operates with a current consumption that is an order of magnitude lower than the other microcontrollers. The MSP430 also has the lowest pin leakage current and

thus wastes very little power. The 6 power modes enable the optimization of low power consumption as required by an application. Furthermore, The MSP430 also features the fastest switching between power modes. It can switch from idle mode to fully synchronized high-speed active mode operation within 6 μs.

The number and type of interrupts available in an MCU also has an effect on current consumption and operational efficiency. More interrupts mean that the CPU does not have to monitor or poll events. Instead, it remains idle until an interrupt brings it into active mode. The MSP430 has at least 40 interrupts. All of its integrated peripherals have interrupts and some have multiple interrupts for different events. Sixteen of its general input-output pins also have interrupt ability to respond to external events.

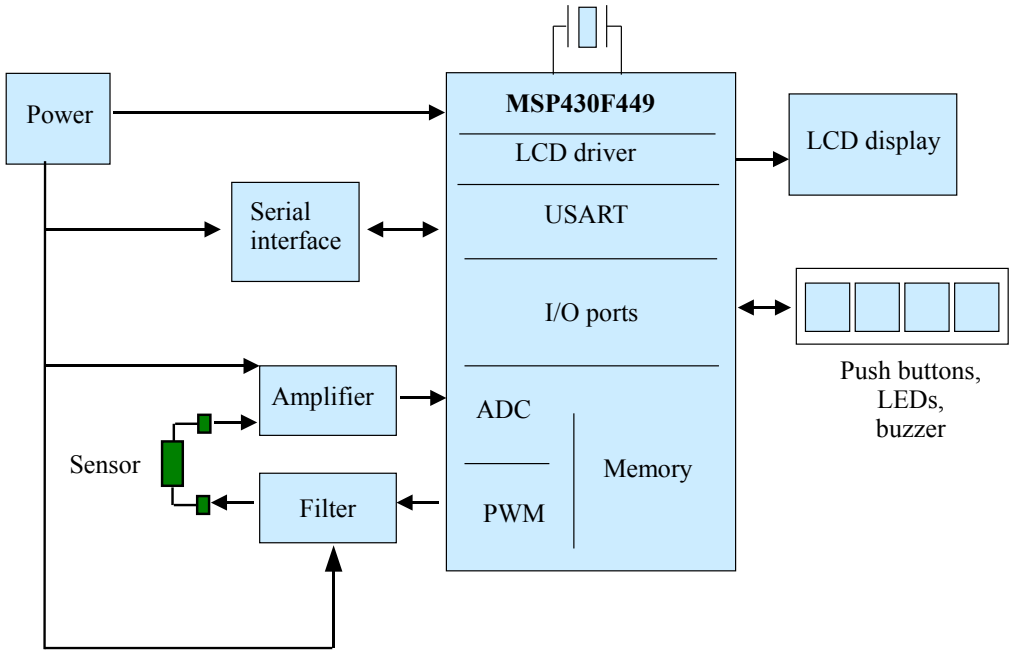


Figure 3.6: Block diagram of the MSP430 based miniEC system. A PWM output from the microcontroller together with the analog filter generates the potential to bias the sensor. The sensor output is converted to a voltage and amplified to a range that can be read by the microcontroller's analog-to-digital converter. The results are displayed on the LCD and a copy stored in the microcontroller's memory for later retrieval via the serial interface.

Figure 3.6 shows a block diagram of the system design using the MSP430. In order to keep the component count low, the microcontroller's internal peripherals were used extensively. Not only does this minimize costs and enable the device to remain compact but it also saves power. The MSP430's peripherals were designed such that they draw no power even when enabled unless they are actively processing data. Of the external peripherals, four push buttons and a liquid crystal display (LCD) form a simple menu driven user interface. The four push buttons labeled stop, enter, up and down are connected to the MCU on port 1. An interrupt is generated every time one of them is pressed. Their functions and effect on the miniEC are defined by software (see section 3). The LCD is the low power 4-mux SBLCDA2 (SoftBaugh, Alpharetta GA) designed specifically to work with MSP430 microcontrollers. It is used to visually confirm device operation, display time of day when the miniEC is in idle mode and display results of running tests. The MSP430 has an on-board LCD driver that powers and drives this LCD directly through pins S0 to S39. On board flash memory is also used for data storage. For this project the MSP430F449's 60 kB memory was partitioned into 40 kB data memory and 20 kB program memory.

An RS-232 serial interface is implemented with the TX and RX pins of the microcontroller's USART module. By default it is programmed to run at 9600 bps using a byte size of 8 bits and 1 stop bit with no parity checking. The serial block is set up using a MAX3221 (Maxim Integrated Products, Sunnyvale, CA) to achieve the necessary level shifting for connection to a PC.[139]

3.2.3 Potentiostat

The amplifier and filter blocks (Figure 3.6) together form a 2-electrode potentiostat. Figure 3.7 shows the schematic of the potentiostat. The circuit was built

with low power, low noise MAX407 op-amps (Maxim Integrated Products, Sunnyvale, CA). Each op-amp (U1A, U1B, U2) draws just 1 μA when idle and about 2 μA otherwise. The MAX407 op-amp has 2 other characteristics, a very high input impedance and a typical low input bias current of 0.01 nA that makes it ideal for potentiostat operations. The IDUA is connected to the output of U1B and the inverting input of U1C. The operation of an op-amp requires that its inputs (labeled + and -) must be at an equal potential. The positive terminal of U2 is connected to ground. This implies that U2's negative input and therefore one set of IDUA electrodes are also at ground potential. The other set of electrodes is maintained at Vbias. No current flows into the op-amp itself, therefore the measured current that flows through the feedback resistor R is the current produced as a result of redox activity.

The excitation potential, Vbias, to the IDUA sensor is created by filtering a PWM signal (generated by timer B of the MCU) through a single pole RC filter. The resulting potential is then buffered and inverted by op-amps before it is connected to the IDUA. The potential produced depends on the value of the PWM duty cycle. A 100% duty cycle represents a potential of 2500 mV. The potential can be changed by the user to the preferred redox potential for the electrochemical species used in the test solution.

The signal from the sensor is a very small current in the nA range. It is, therefore, necessary to amplify and convert it to a voltage range (0-2500mV) that the MCU's analog-to-digital converter can measure. This is achieved with the transimpedance amplifier circuit (Figure 3.7). Based on Ohm's law the amplifier gain is determined by the value of the resistance R in the feedback loop of the op-amp U1C. With an R of 2.0 MΩ a current of 1 nA is converted to a voltage of 2 mV, so the miniEC is able to measure a wide current range from 0 to 1250 nA.

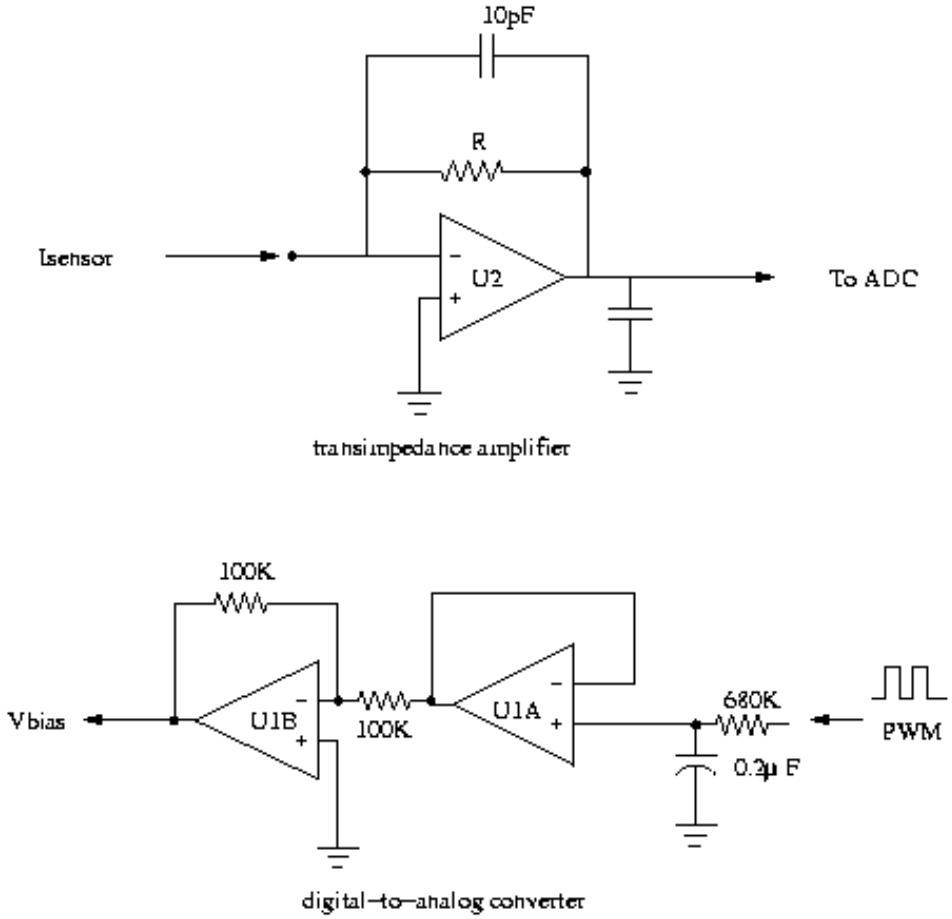


Figure 3.7: Schematic of the analog block showing the miniEC's two electrode potentiostat set-up. The sensor potential is generated with a filtered PWM signal from the microcontroller. The sensor current is measured with a transimpedance amplifier and sent to the microcontroller. The 10 pF capacitor reduces the intrinsic noise of the resistance R

In addition, the amplifier can be made programmable by replacing R with a range of resistors and a digitally controlled switch, such as the MAX4652, for variable gain selection.

3.2.4 Power supply

The miniEC can be run directly off of a battery pack since the miniEC's low power components operate within a range (1.8 to 3.6V) that coincides with the useful life of a 2 AA, 2 AAA battery pack or a single 3V cell. Below a supply voltage of 2.7V however, the MSP430 microcontroller cannot write data to memory. Managing the battery voltage with a voltage regulator compensates for linear battery discharge profiles and provides a constant supply voltage to the miniEC circuitry regardless of varying current demands. Texas Instrument's TPS60310 high efficiency charge pump was selected for this purpose.

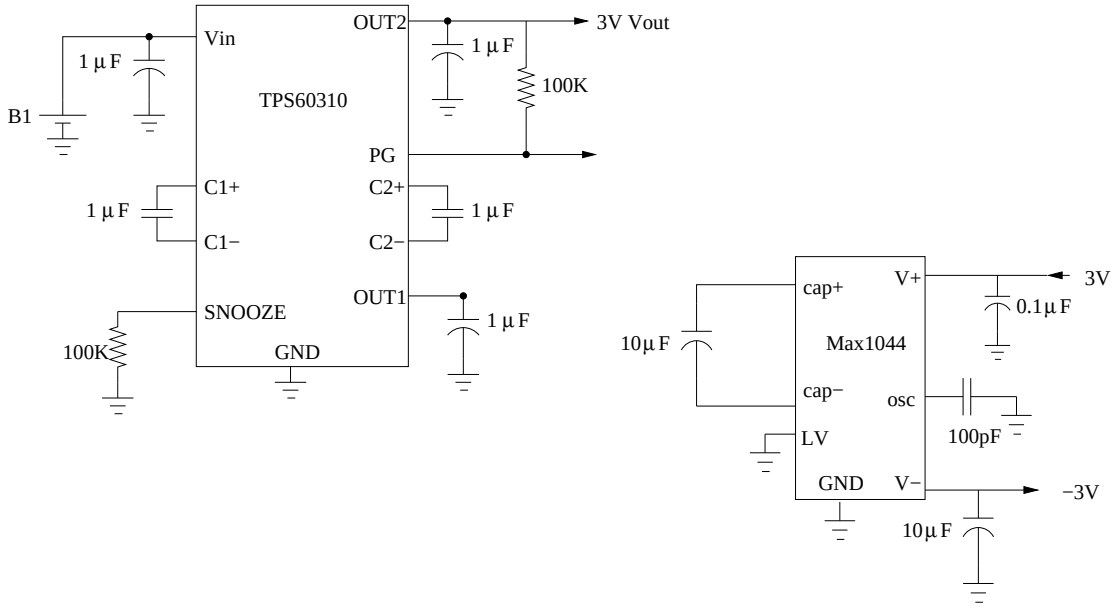


Figure 3.8: The TPS60310 converts the input from a single AA or AAA cell (B1) to a regulated 3V system supply. The 3V powers a MAX1044 that produces a symmetrical negative supply.

The TPS60310 produces a regulated 3V output voltage from a 0.9 to 1.8V input thus reducing the battery requirement to a single AA or AAA battery cell (Figure 3.8). In 'snooze mode' it has a quiescent current of just 2 A but can supply 2 mA to the rest of the circuit. 2 mA is enough for most of the miniEC's operation. The MCU may require more than 2 mA momentarily for certain events such as writing

data to flash memory. Demand for more than the 2 mA threshold, causes the TPS60310 to automatically exit snooze mode to supply up to 20mA. The snooze input of the TPS60310 is tied to the ground potential (logic low) which forces it to return to snooze mode after the high demand event. A MAX1044 (Maxim Integrated Products, Sunnyvale, CA) inverter is used to produce a negative supply -Vcc for the digital-to-analog filter circuitry since both a positive and negative power supply is needed for the op-amps in the circuit that converts the PWM signal to the IDUA bias potential (Figure 3.7).

3.3 Software

Given the pace of innovation in microelectronics, it was expected that newer, faster and better MCUs would be produced within each family of microcontrollers after the initial selection process. Therefore, to ensure portability of firmware to new MSP430 MCUs or a different MCU family, the embedded software was written in C using the free mspgcc tool chain.

The firmware logic is event driven. The MCU is programmed to run in low power mode 3 (LPM3 or idle mode) until an event puts it into active mode by waking up the CPU (Figure 3.9). The MCU returns to the idle mode as soon as it completes the action associated with the event.

On startup, the MCU is programmed to initialize program variables such as the location of data memory, the time and the default values. The basic timer serves as the pulse for a real time clock so it is set up to cause an interrupt every second. Port 1 is set up as interruptible input for the button interface. Port 2 and Timer B are setup for PWM output. LCD control, ADC, and the USART are enabled as well. After setup, all interrupts are enabled. The MCU then proceeds into the main routine loop that logically implements a finite state machine. Figure 3.10

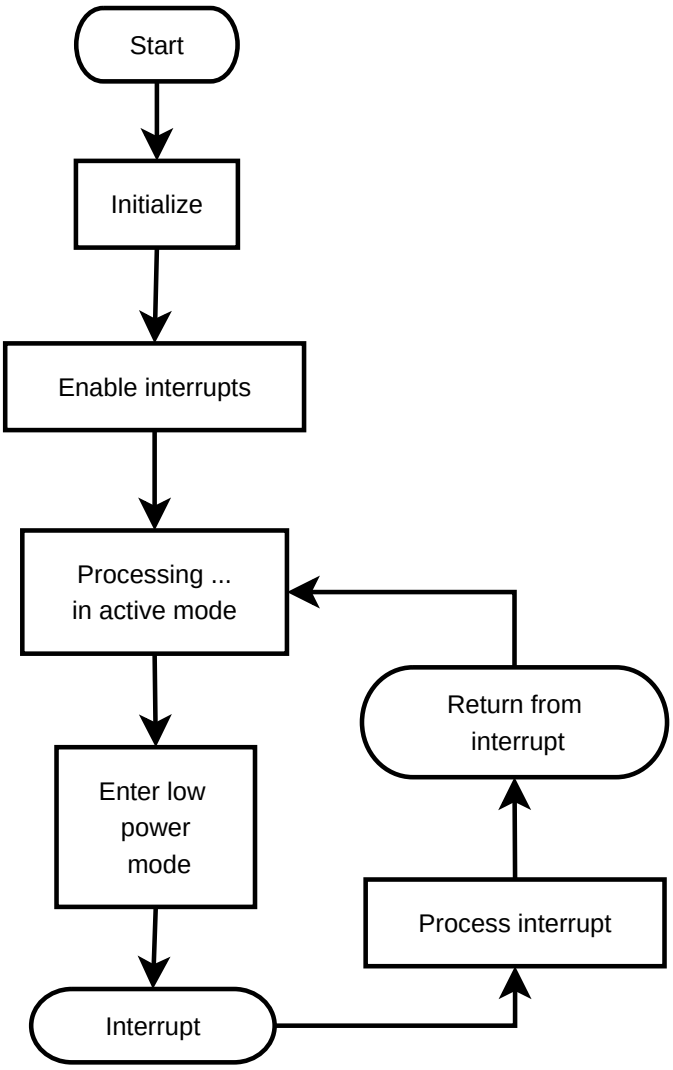


Figure 3.9: Microcontroller firmware logic flow. Operation is interrupt driven. The microcontroller stays idle until an interrupt occurs. It then enters active mode and performs the action requested by the interrupt

shows the relationship between the states and the events that cause the miniEC to transition from one state to another. The user operates the device by pressing one of the 4 buttons or by sending commands mimicking button events via the serial connection.

The main loop starts out in the IDLE state. In this state, all peripherals except the basic timer are powered down. Time is updated and displayed on the LCD. Pressing the 'enter' button sends the system into the READY state. The MCU

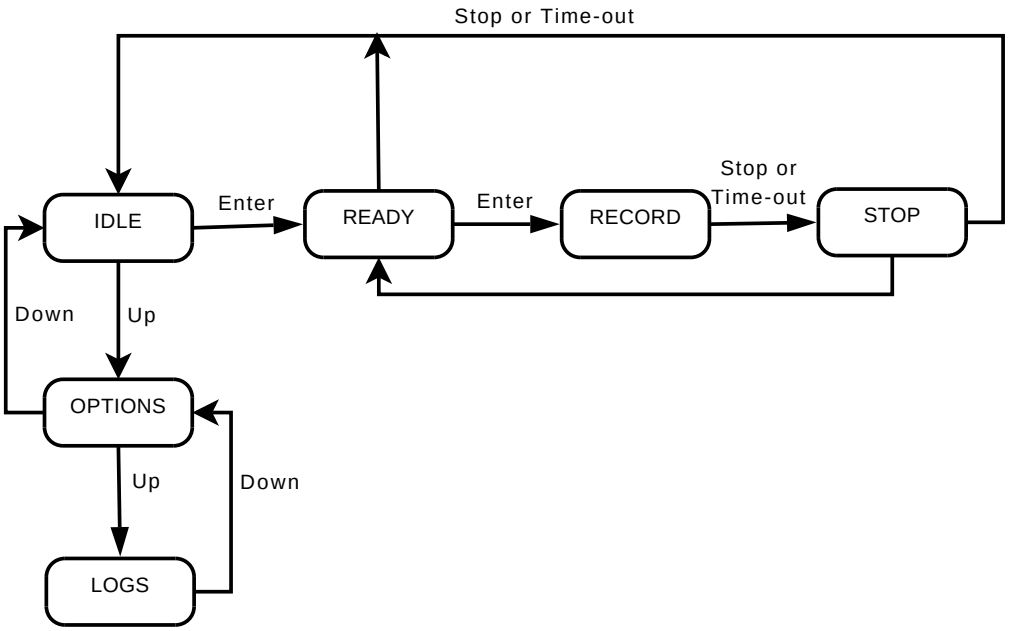


Figure 3.10: MiniEC main loop state chart. The device starts out in the idle state. When the 'enter' button is pressed, the miniEC transitions to the ready state. In the ready state another 'enter' button event transitions the miniEC into record state. During recording, the test can be stopped by pressing the 'stop' button. In stop mode, pressing the 'enter' button will start a new recording. The stop and ready modes have timeout periods. If no events occur within the timeout periods, the miniEC will return to the idle mode. The user reverts to the idle state from other states by pressing the 'stop' button.

enables all the necessary peripherals (PWM, ADC, USART, LCD controller) and checks that they are operating properly. It also determines if enough memory to store the data is available. If everything passes these controls, the message "Ready" is displayed on the screen. Recording starts when the user presses the 'enter' button again. At the start of the recording session, the time is noted and a file is opened to store the results for the duration. At each interval, the recorded data is displayed on the LCD and a copy is stored (or transferred to a PC). At the end of the measurement duration or if a user presses the 'stop' button, the miniEC closes the data file and enters the STOP state. From this state, the user can initiate another measurement or return to the IDLE state. In both the READY and STOP states, the system will revert to the IDLE state after 5 minutes of inactivity.

In the OPTIONS state the user can cycle through variables to update time, and change the settings for electrochemical measurements. By default, the MCU is programmed to perform direct current amperometry (DCPA) measurements. The potential range available for DCPA is from 0 to 2500mV in steps of 10mV. The interval range available is from 1 to 60s in steps of 1s. The duration can be set from 1 minute to 24 hours. The amount of data that can be stored is limited by the interval selected. At 1-minute intervals 10 days worth of data can be stored, while at 1-second intervals only about 4.5 hours of data can be stored. Once the data memory of the miniEC is full, measurements can still be made and viewed or transmitted. The miniEC will not automatically delete any files. Instead the user is notified of the memory full condition on the LCD and prompted to delete files when convenient. Each recorded file is assigned a numerical ID. In the LOGS state, the user can delete files or use the up/down buttons to cycle through the files to view the value of the peak signal recorded for each one.

A variety of strategies are used in the firmware to achieve low power operation and to optimize performance. An example is implementation of a software filter by oversampling and then averaging the sensor signal. For each interval, the ADC takes 256 (4^4) successive readings and records the average value. Each factor of 4 adds 1 bit of resolution to the measurement and improves the signal-to-noise ratio by a factor of 6 db.[140] As a result, this simple technique greatly improves the signal-to-noise ratio and effectively increases the ADC resolution from 12 to 16 bits without adding extra components and expense. Furthermore, the CPU is disabled during sampling and conversion of signals by the ADC. Fast processing speed is maintained by keeping the system in LPM0, an intermediate low power mode between idle mode and active mode that uses less than a fifth of the active mode power. The MCU is fully active and thus consuming the most power mainly

for the few milliseconds it takes each interval to average, scale and store the sensor data.

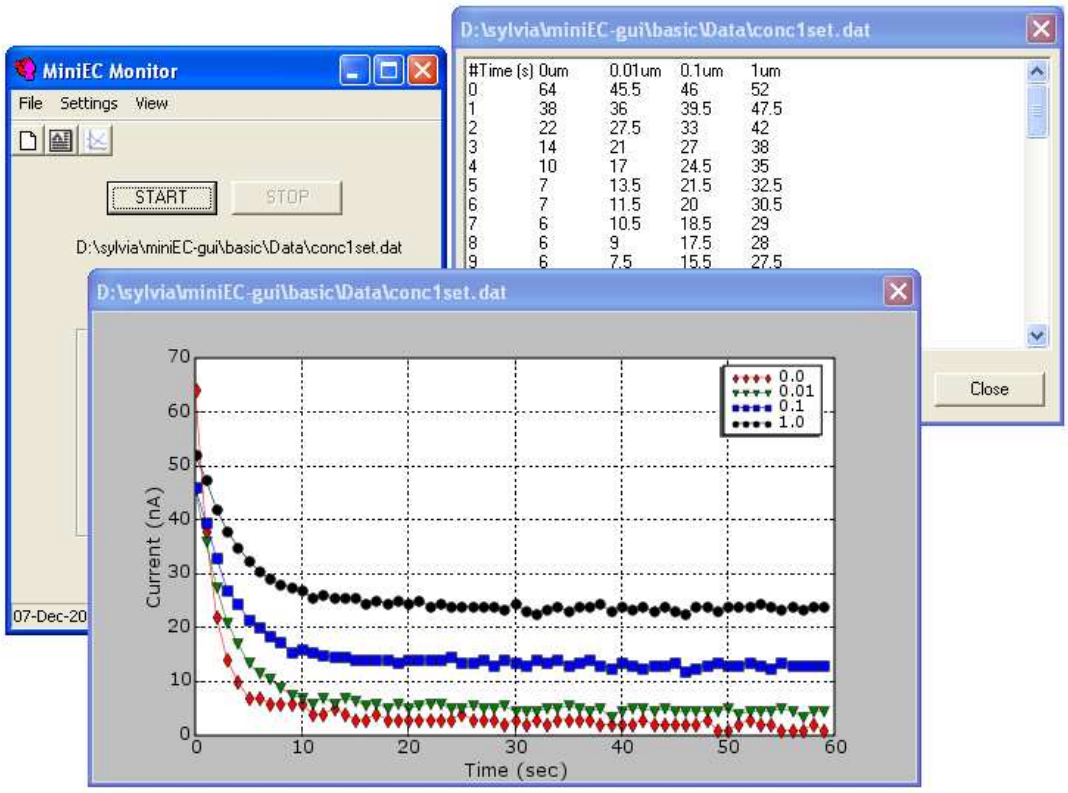


Figure 3.11: A screenshot of the miniEC graphical user interface displaying sensor current over time for potassium ferri/ferrohexacyanide concentrations of 0, 0.01. 0.1 and 1 M in static 10 μL droplets.

The MSP430 is programmed to transmit serial data in human readable ASCII format. This makes the output easy to view and capture with any terminal program such as HyperTerminal (on Microsoft Windows), ZTerm (on MacOs) or Minicom (on Linux/Unix). The provision of a graphical user interface (Figure 3.11) however simplifies communication with the miniEC and provides opportunity for real-time visual monitoring and graphing of data trends. In addition a user can perform system setup with a familiar user-friendly interface. The miniEC system thus includes a small cross platform interface implemented with the royalty-free WxPython toolkit.

3.4 Discussion

The miniEC (Figure 3.12) is intended for electrochemical measurements by technical and non-technical users. As such its use does not require any special skills. Its 4-button keypad and LCD provides an easy to use interface (Figure 3.12). By default, it is set to perform constant potential amperometric experiments but can easily be programmed to perform other electrochemical tests. With test settings of 400 mV potential, test duration of 1 minute and sampling interval of 1-second, the estimated average current consumption is 2.5 mA. At this rate, the miniEC can run continuously for 2 weeks taking data at 1-second intervals with a single AAA battery rated at 1000 mAh. In a typical scenario where the miniEC is used intermittently, for example, a total time of 2 hours a day in the default configuration, it can run at least 6 months on a single AAA battery. Most of the electronic parts selected for the miniEC are rated to perform in the industrial temperature range of -40 to 85 C. Thus they work with minimal deviation within the range of 0 to 70C typical for mass market commercial devices without extra temperature compensation circuitry. The parts along with their prices in prototype quantities are listed in table 3.2.

Table 3.2: Bill of materials for the main components of the miniEC instrumentation

Component	Quantity	Description	Unit price	Cost
TPS60310	1	Charge Pump Converter	$2.36	$2.36
MAX3221	1	RS232 DRIVER	$1.38	$1.38
MAX406	1	Current feedback op-amp	$2.17	$2.17
MAX407	1	Dual current feedback op-amp	$4.50	$4.87
MAX1044	1	Negative voltage converter	$2.73	$2.73
MSP430F449	1	Mixed Signal Microcontroller	$8.00	$8.00
SBLCDA2	1	3V liquid crystal display	$4.50	$4.50
DB9	1	Sub DB9 female connector	$0.40	$0.40
Other	–	Resistors, capacitors, LED etc.	–	$10.00
		Total component cost		$36.41

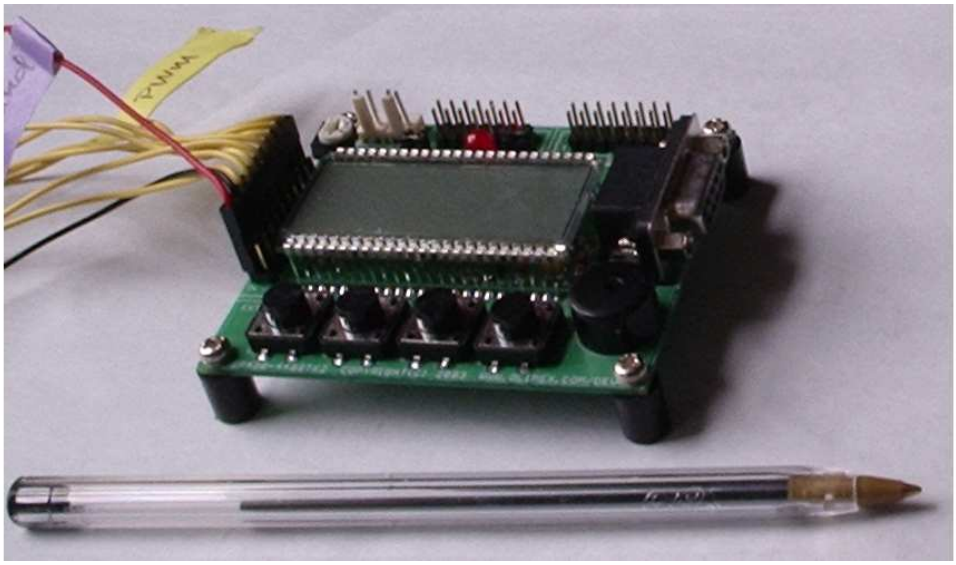


Figure 3.12: Photograph of prototype miniEC instrumentation (assembled by Olimex Ltd., Plovdiv, Bulgaria). The overall dimensions are about 80 by 65 mm. The pen is shown for size comparison.

Subsequent to the current design and implementation, Texas Instruments released the MSP430FG439 designed specifically for portable medical applications. The MSP430FG439 has 3 integrated programmable gain op-amps to take analog readings and a 12-bit DAC to generate potentials from 0 to 2500mV. It also supports direct memory access so data can be manipulated without the intervention of the CPU. Apart from these enhancements, it is identical to the MSP430F449. The microcontroller firmware described above was successfully ported to the new chip to take advantage of the new peripherals, thus, validating our approach to miniEC instrumentation development. This approach will enable the production of a range of devices evolving from the basic miniEC functionality described here. Devices can be made smaller or functionality can be added to create deluxe versions with more features. Even with the MSP430F449, there is still plenty of room for incremental enhancements which is currently under investigation. The ADC port, for instance, has 8 inputs, thus, with the addition of sensor hardware and minor modifications to the embedded software, the miniEC can monitor up to 8

samples at a time. Storage capacity can also be expanded with serial memory chips via the microcontroller's serial peripheral interface.

The system's flexibility is not limited to the hardware architecture. The finite state machine main loop algorithm for the miniEC is a very efficient way for implementing control logic. The FSM code uses status bytes and a fast lookup table to minimize CPU overhead. The system's behavior is encapsulated by states and actions with well defined conditions for state transitions. Given a set of inputs and a known current state, the state transition can be predicted, allowing for easy system simulation and testing. The technique also encourages modular design thus simplifies firmware upgrades in the future.

The accuracy of the miniEC was evaluated by comparing its performance to an industry standard bench top electrochemical workstation in static and dynamic constant-potential amperometric detections of the redox couple potassium ferri- and ferrohexacyanide.[133] In both sets of experiments, the inexpensive miniEC performance was comparable to that of the electrochemical workstation. In fact, the miniEC achieved a detection limit of 0.01 μM combined ferri/ferrohexacyanide concentration which was 10x lower than that of the standard lab-bench system. Experiments were done in static mode (i.e. stationary droplets) and dynamic mode (i.e. flowing in a microfluidic channel). The response time of the miniEC was in all cases very similar to the exquisite response time of the bench top electrochemical workstation.

In static mode, typically better percent errors were obtained with the miniEC system (except of for the signals of pure buffer which were 1.8$\pm$0.5 and 2.0$\pm$0.5 nA for the miniEC and bench top potentiostat, respectively) and were between 11 and 2% depending on the concentration detected (between 0.01 and 50 μM) with the miniEC, and between 19 and 7% with the bench top potentiostat. In the case

of the dynamic experiments, percent errors were slightly better with the bench top system (between 0.6 and 3.8%) than with the miniEC (between 1 and 6%). It was therefore concluded that the miniEC was a useful transducer and detector system for a complete biosensor assay as described in Chapter 4 for the detection of Dengue virus RNA.

3.5 Conclusion

We have presented the miniEC, an electronic appliance designed as a portable electrochemical transduction system with integrated recording and data manipulation. It potentially could replace electrodes, potentiostat and computer systems in conventional electrochemical set ups.

Use of the inexpensive, low-power, powerful and efficient MSP430 microcontroller together with other carefully selected low power components results in a design that can run for months on a single AA or AAA battery. Also, emphasis on low cost resulted in a compact design with a minimal number of off-the-shelf components that altogether cost less than $50 in prototype quantities.

Both the hardware and software designs are general, modular and flexible. The microcontroller can be programmed to perform different electrochemical tests providing the user with a fully integrated mini-electrochemical system. The application described here in a constant-potential amperometry configuration used only a fraction of the MCU's capability. The sophisticated MSP430 can also control many additional components (a pump actuator for instance) and could perform other functions such as digital signal processing of complex electrochemical signals in future prototypes.

CHAPTER 4

ELECTROCHEMICAL MICROFLUIDIC BIOSENSOR WITH INTEGRATED MINIPOTENTIOSTAT[†]

4.1 Introduction

Microfluidics is the enabling technology base for the development of miniature devices that move, mix, control and react with fluids volumes in the micron range.[141–143] Microfluidics offer obvious advantages in the reduced consumption of reagents: faster and more sensitive reactions due to enhanced effects of processes such as diffusion and mass transport; increased throughput through parallel processing; and reduced expenses in terms of power and reagent consumption. Most importantly, fabrication of microfluidic devices can be inexpensive and allows the integration of several modules to automate analytical processes.[19,144] Thus, this technology has the potential to answer the heightened need for inexpensive, sensitive, rapid and portable detection systems following recent threats of bioterrorism. To this end, new integrated microfluidic-based techniques have been developed recently such as plug-based chemical screening,[145,146] which combines microfluidics with a heterogeneous immunoassay to detect bacterial toxins.[147] Others have used optics,[148] electrowetting[149] and electrophoresis[150] with microfluidics to detect other target molecules. Most of the innovation has been achieved with the integration of molecular biology-based technologies into microfluidic devices, such as the polymerase chain reaction.[151,152] However, while sensitive and rapid detection is feasible with many systems, decreasing analysis cost and portability has not been so easy since in most cases optical detection requiring complex optics for signal

[†]A VERSION OF THIS CHAPTER WAS PUBLISHED AS "ELECTROCHEMICAL MICROFLUIDIC BIOSENSOR FOR NUCLEIC ACID DETECTION WITH INTEGRATED MINIPOTENTIOSTAT." IN THE JOURNAL BIOSENSORS AND BIOELECTRONICS, VOLUME 21, NUMBER 12, PAGES 2217–2223

quantification has been chosen.[96,97,153,154]

We previously presented a microfluidic biosensor combined with an optical liposome signal amplification scheme as a promising inexpensive solution to the heightened need for technology that can rapidly and accurately detect pathogenic organisms in environmental, clinical and food samples.[96,97] Liposome technology has previously been used in analogous membrane detection systems with great success.[50,155,156] Our conversion of the system to a microfluidic format significantly reduced reagent use and produced a 100-fold increase in sensitivity over our analogous polyethersulfone membrane system. Esch et al. have also reported gains in sensitivity by converting a liposome based membrane detection assay for Cryptosporidium parvum to a microfluidic format.[51] These microfluidic systems relied on fluorescence microscope detection of the capture zone in which sulforhodamine B encapsulating liposomes accumulated directly proportional to the amount of target nucleic acid sequence present.

In general, optical and fluorescence-based transducers require an expensive microscope equipped with fluorescence filters, other electronics and a computer to measure and quantify the fluorescent signal if low detection limits are desired.[153] In contrast, equipment for electrochemical sensor-transducer can be simpler, less expensive and portable.[157] In fact, miniaturized amperometric sensors that combine the specificity and selectivity of biological components with the analytical advantages of electrochemical detection have been reported by several groups monitoring enzymatic activity.[158,159] Similarly, most commercially available glucose sensors rely on electrochemical detection.[160]

By combining microelectronics and microfluidics with the simple and effective liposome signal enhancement technology and an amperometric transducer, we have designed a miniaturized electrochemical detection system (miniEC) that is easy to

assemble and use. Physically, the miniEC has two parts. It consists of a potentially disposable microfluidic sensor cartridge and the supporting instrumentation to power, measure and display the sensor results. The amperometric transducer, an interdigitated ultramicroelectrode array (IDUA), was placed directly into the microchannel of the cartridge. DNA and RNA molecules are quantified in this device based on a sandwich hybridization assay similar to the one previously described[96],[97] with the transduction mechanism being based on electrochemical rather than fluorescence detection.

4.2 Experimental

4.2.1 Materials and reagents

General laboratory chemicals and buffer reagents were purchased from VWR Scientific Products, (New York, NY, USA) and Sigma Chemical Company. Lipids were obtained from Avanti Polar Lipids (Alabaster, AL, USA). Sulforhodamine B and Streptavidin were purchased from Molecular Probes Company (Eugene, OR, USA). Oligonucleotides were synthesized by the BioResource Center, Cornell University (Ithaca, NY, USA). Flexible tubing and syringes were also purchased from VWR Scientific Products. Sylgard-186 PDMS elastomer kit and Dow corning 1593 adhesion promoter were purchased from Wolcott-Park Inc.(Rochester, NY, USA). Superparamagnetic beads (MyOne streptavidin) were purchased from Dynal Biotech Inc. (Lake Success, NY, USA) and magnets from Magnet Applications (Horsham, PA, USA). Tube connectors were supplied by Small Parts Inc., Miami Lakes, FL. Plexiglas was purchased from Ithaca Plastics (Ithaca, NY, USA). Unless specified, all electronic components were purchased from Digi-Key Corp. (Thief River Falls, MN). The Cornell Nanofabrication Facility (CNF) provided

clean room facilities, chemicals and equipment for silicon template and microelectrode fabrication. Electronic circuitry was assembled by Olimex Ltd., Plovdiv, Bulgaria.

4.2.2 Interdigitated ultramicroelectrode array fabrication

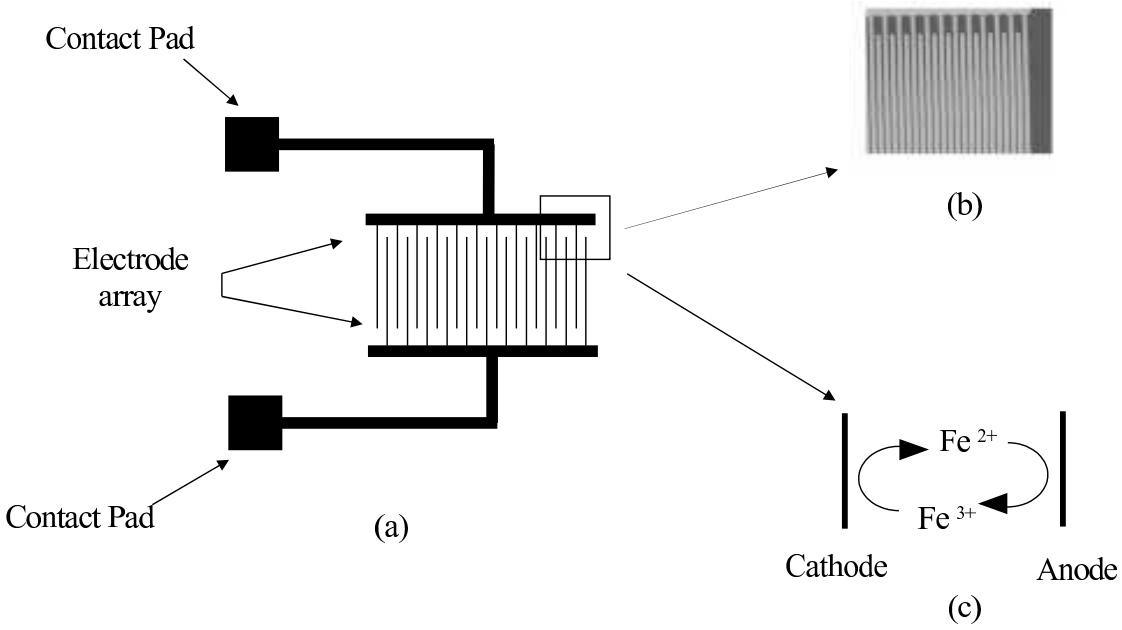


Figure 4.1: Schematic, optical micrograph and electrochemical reactions of an interdigitated ultra microelectrode array (IDUA). (a) Schematic diagram of an IDUA, (b) Optical micrograph of a gold IDUA, (c) illustration of cycling of oxidation and reduction of potassium ferrihexacyanide (Fe^{2+}) and potassium ferrocyanide (Fe^{3+}) in the gap between a pair of electrodes.

The interdigitated ultramicroelectrode arrays (Figure 4.1) were fabricated by standard clean-room photolithographic and lift-off techniques on glass wafers as previously described.[119,137] Briefly, a mask was prepared using a Pattern Generator that was 5x the final size. Using a g-line stepper (436 nm) the pattern was transferred to Pyrex glass wafers (Corning 7740, Corning, NY) coated with positive photoresist (Shipley S1813). Subsequently 10 nm titanium and 150 nm gold were evaporated using an e-gun source (CVC 4500 Evaporator). The wafers were then

soaked in acetone to lift off the excess metal and photoresist leaving the patterned electrode arrays. Before use, the glass wafers were rinsed with isopropanol followed by deionized water, dried and then diced into single IDUA chips. The resulting IDUAs had 500 pairs of fingers with widths of 3.8 μm and gap sizes of 2.5 μm.

4.2.3 Microchannel fabrication and preparation

Figure 4.2 shows the layout for the microchannel device used in this study. Standard photolithography and dry etching techniques were used to produce a positive-relief image of the microchannels on a 4-inch silicon wafer as described previously[96,97] . Using soft-lithography, the microchannels were realized in polydimethyl siloxane, (PDMS) covered with a glass slide bearing the IDUA and packaged in a Plexiglas housing.[97]

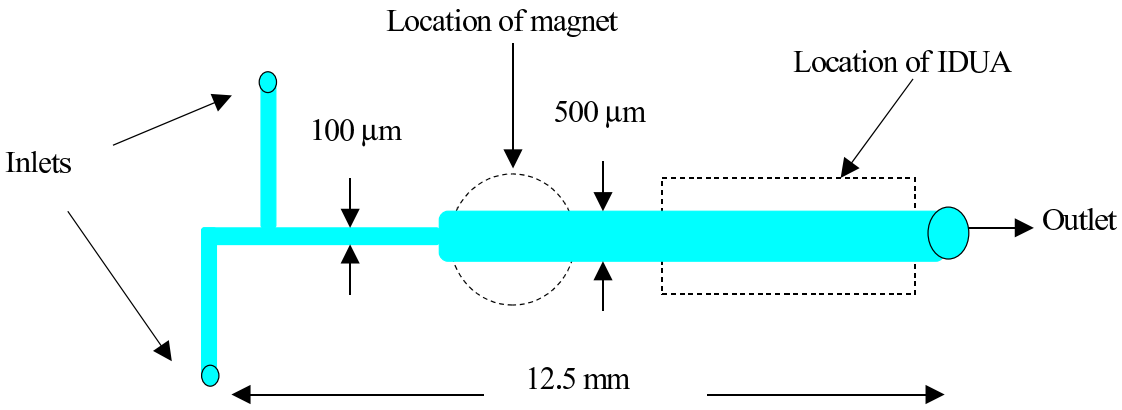


Figure 4.2: Layout and dimensions of the microchannel device for electrochemical detection. The microchannel has 2 input and 1 output port. The depth of all channels is 50 μm. The enlarged 500 μm wide region coincides with the areas of immobilization (magnet) and detection (IDUA).

The PDMS microchannels and IDUA glass slide were placed between 2 Plexiglass plates with the IDUA chip below and the microchannels above. The top Plexiglass plate had plastic tubing glued into locations that lined up with the inlet and outlet ports of the microchannel. Another hole was drilled into the upper

plexiglass plate to accommodate the permanent magnet (35DNE1304-NI) needed for the immobilization of the superparamagnetic beads. Eight screws were used to tighten the two Plexiglass plates together and to provide enough pressure for leak tight operation of the assembled microfluidic device.

4.2.4 Miniaturized electrochemical detection unit

A minipotentiostat, the miniEC, was developed that could provide the electrochemical transducer, the IDUA, with a constant potential, report the current signals produced during the redox reactions and display the signal on a digital display. The electronic circuitry of the miniEC was powered by 1 AAA battery. A block diagram is shown in Figure 4.3.

The heart of the instrumentation is an MSP430FG439 signal-chain-on-chip microcontroller (Texas Instruments, Dallas, Texas) which integrates numerous functions that traditionally have been executed with discrete chips. The microcontroller was programmed to perform direct current potential amperometric (DCPA) experiments with the free GNU C compiler tool chain, mspgcc. In DCPA, the digital-to-analog converter (DAC) peripheral of the microcontroller supplies the potential difference for the IDUA. In the presence of electrochemically active compounds, the current produced by the electrode is first converted to a voltage, amplified by an op-amp (MAX418, Maxim Integrated Products, Sunnyvale, CA) in a simple current-to-voltage configuration and then sent back to the microcontroller's analog-to-digital converter (ADC) module. The microcontroller displays the result on the liquid crystal display (LCD) and stores a copy of it in memory as a digital value for later retrieval. Retrieval is achieved via an RS232 serial connector controlled by the microcontroller's universal synchronous asynchronous receiver transmitter (USART) peripheral. The 4 pushbuttons allow a user to start and

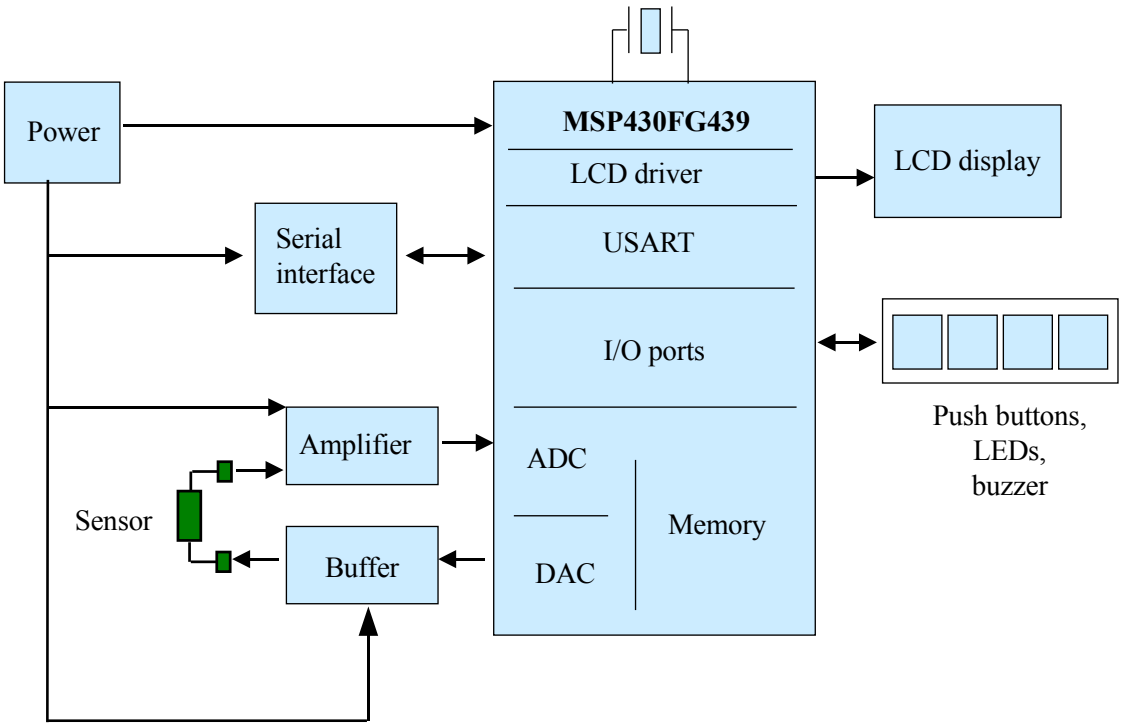


Figure 4.3: Block diagram of miniaturized electrochemical detection instrumentation designed for portable detection. Most of the system functionality is provided by the MSP430 microcontroller, enabling the device to remain compact. The digital-to-analog converter (DAC) generates the potential to bias the sensor. The sensor output is converted to a voltage and amplified to a range that can be read by the analog-to-digital converter (ADC). The results are displayed on the liquid crystal display (LCD) and stored in the microcontroller memory for later retrieval via the serial connection. The whole system is powered by 1 AAA battery.

stop an experiment or to change the settings for the potential difference and duration of measurement. The overall dimensions of the device are about 80 by 65 mm.

4.2.5 Electrochemical measurements

For testing of the miniEC unit, a potential of 400 mV was applied across the IDUA. Current generated by the combined molarity of 0.01 to 100 μM of equimolar concentrations of potassium ferrohexacyanide and potassium ferrihexacyanide (ferro/ferrihexacyanide) in 0.1 M potassium phosphate buffer (pH 7) were mea-

sured. The miniEC performance was evaluated first in a static context with 10 μL droplets of ferro/ferrihexacyanide solution on the electrode array. Measurements were taken by the miniEC for a duration of 60 s at 1 s intervals. Between experiments, the electrode was rinsed with 20 μL 0.1 M potassium phosphate buffer.

For the evaluation of the miniEC in a dynamic context, experiments were performed with the IDUA assembled inside the PDMS microfluidic cartridge. Varying volumes (50, 100 and 200 nL) of ferro/ferrihexacyanide were injected at a flow rate of 10 μL/min into a continuous buffer stream flowing across the IDUA at a rate of 1 μL/min.

Using the same microfluidic sensor cartridge and solutions all measurements were repeated with the Epsilon Electrochemical Workstation (Bioanalytical Systems, West Lafayette, IN) to evaluate the performance of the miniEC.

4.2.6 Detection of specific RNA sequences

RNA derived from nucleic acid sequence based amplification of a segment of Dengue virus 3 RNA was used as model analyte. Reporter and capture probe sequences are given in Table 4.1.

Table 4.1: DNA sequences of the Dengue virus serotype 3 probes

Name	Sequence (5'-3')
Reporter probe	gAT gCA Agg TCg CAT Atg Ag
Capture probe	Agg gAA gCT gTA CCT CCT TgC AAA g

The testing protocol was adapted from the fluorescence detection approach published earlier[97,98] to allow for electrochemical detection. In brief, 1 μg of capture probe coupled magnetic beads, 2 μL of reporter probe coupled liposomes encapsulating 150 mM ferro/ferrihexacyande, 2 μL of hybridization buffer (60% formamide, 6xSSC, 0.8% Ficoll type 400, 0.01% Triton X-100, 0.15 M sucrose) and

2 μL of target RNA (amplified Dengue virus serotype 3) or deionized water (control) are incubated at room temperature for 15 minutes and then injected into the microfluidic biosensor at a flow rate of 1 μL/min. The beads with bound target and liposomes are captured at the magnet. Unbound liposomes and RNA are washed out with 10 μL of running buffer (10% formamide, 3xSSC, 0.2% Ficoll type 400, 0.01% Triton X) followed by liposome lysis with 25 mM β-octylglucopyranoside (OG). The redox cycling of the released ferro/ferrihexacyanide on the IDUA in the biosensor is subsequently measured by the miniEC.

4.3 Results and discussion

The integration of an electrochemical microfluidic biosensor with a minipotentiostat and microcontroller was investigated obtaining a miniEC system. The electrochemical detection was based on redox cycling of potassium ferro/ferrihexacyanide by an IDUA as described earlier.[119] This two-electrode set up was assumed to be an ideal system for a microfluidic sensor since no external reference electrode was required, a simple circuitry could provide reliable signal data and very low detection limits could be obtained. MiniEC requirements for portability, low power consumption and a small form factor were achieved with an electronic design that used low power electronic components and as few components as possible. The overall system consisted of the IDUA and instrumentation that included a microcontroller unit, a current-to-voltage converter, an LCD, a pushbutton interface for user input and a serial connection for networking. More detail about the electrical circuitry, firmware and software developed can be found in chapters 3, 5 and 6 respectively.

4.3.1 Static electrochemical experiments

The electrochemical performance, signal generation and data output of the miniEC was compared to IDUAs connected to a standard lab-bench electrochemical work station, the BAS Epsilon. 10 μL droplets of the solutions were placed on top of the IDUA and data were collected at 1 s intervals with a potential of 400 mV applied for a duration of 60 s. Figure 4.4 shows the typical profiles of IDUA current output recorded by both the miniEC and the Epsilon for high (over 10 μM) and low concentrations of potassium ferri/ferrohexacycanide. Table 4.2 summarizes the test results showing the average of steady-state signals, initial slope and the time to reach steady state for both systems.

At low concentrations, the traces measured with both device set ups were similar (Figure 4.4). However, with lower background signals and a higher specific signal obtained through the miniEC, the signal to noise ratios were higher for the miniEC at low concentrations (Figure 4.5a) making the miniEC the more sensitive device. At higher concentrations (Figure 4.4), however, the miniEC signal was lower and device response was slower in reaching steady state, i.e. requiring 5 s or more vs. an almost immediate steady state in the case of the Epsilon set up. The signal to noise ratios at higher concentrations (Figure 4.5b) were not significantly different.

A dose response curve of the steady state signals of the miniEC was plotted (Figure 4.6). The background signal obtained when analyzing buffer solution only was 1.8 nA and was subtracted from all data points. A limit of detection of at least 0.01 μM of potassium ferri/ferrohexacyanide was determined. The signal obtained, 4.5 nA, was more than the signal of the buffer solution plus three times the standard deviation (1.8 nA + 3*0.5 = 3.3 nA). Similarly, a dose response curve was obtained using the Epsilon set up with which a limit of detection of 0.1 μM

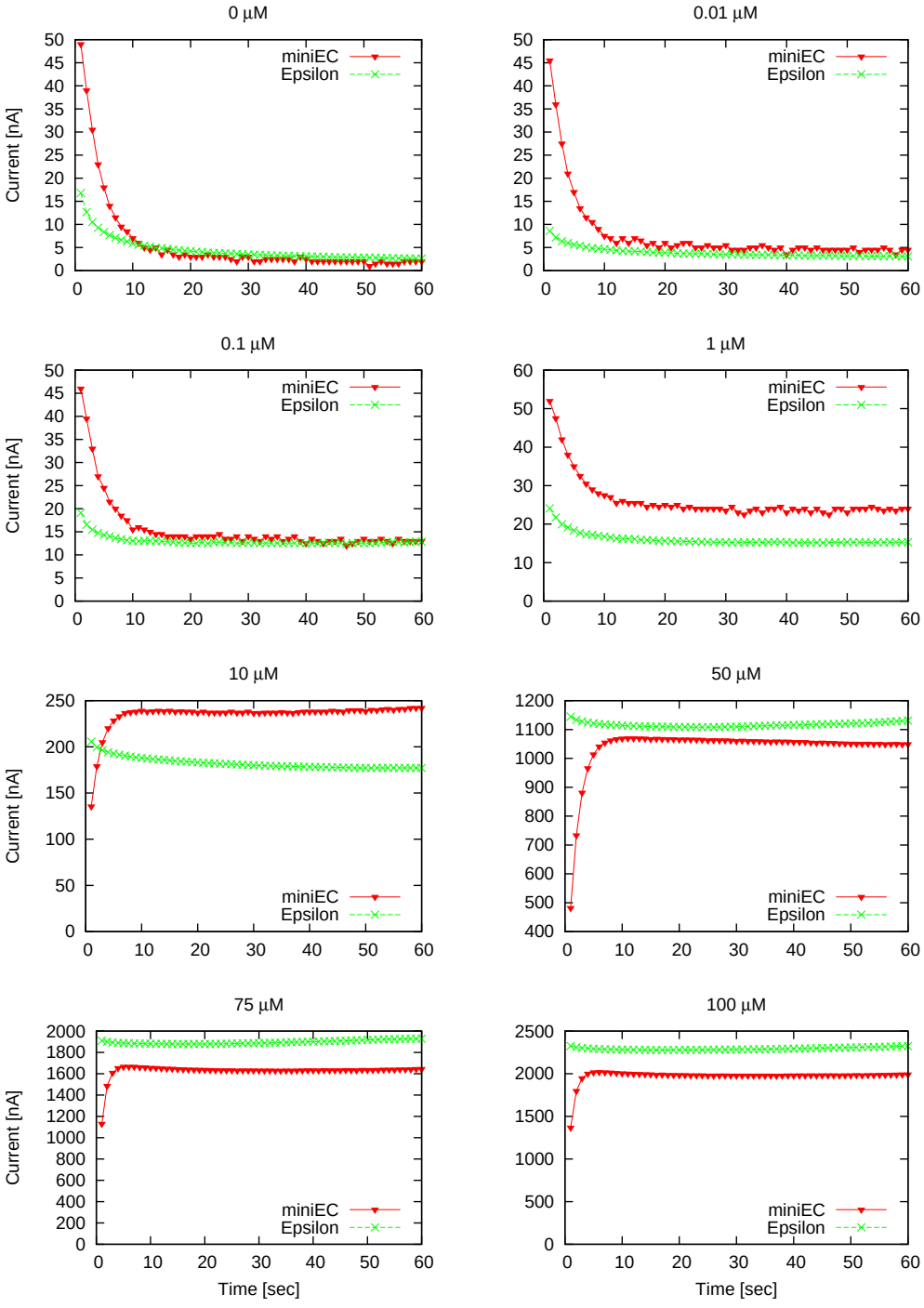
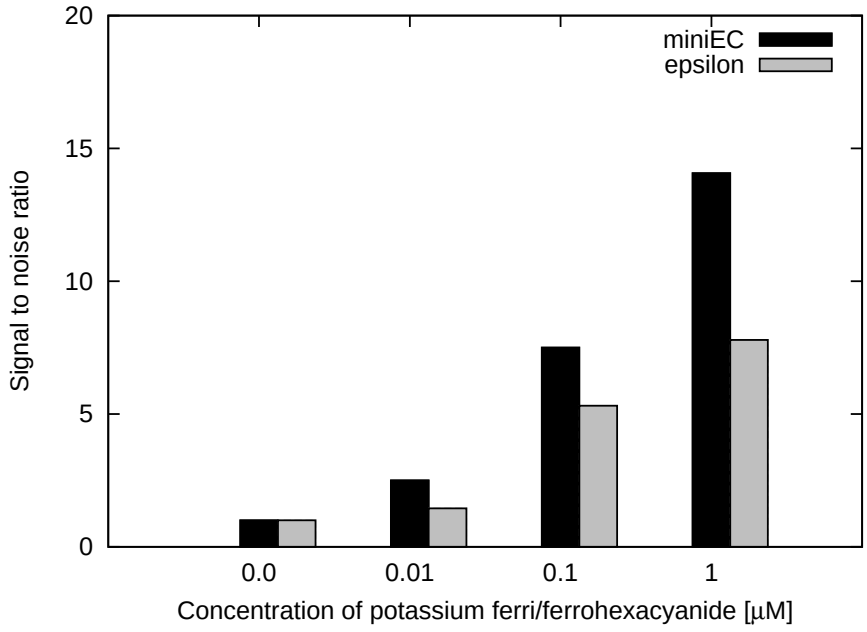


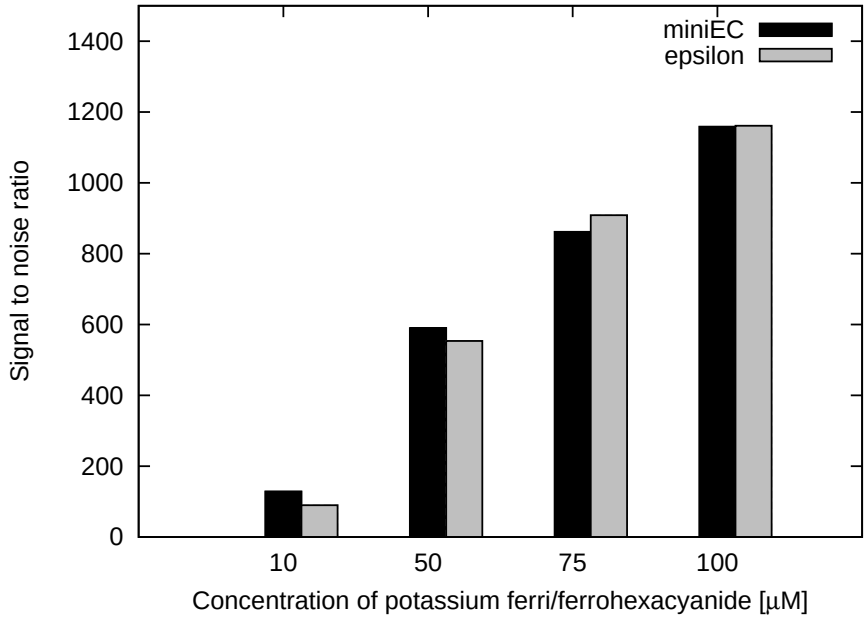
Figure 4.4: Profile of the IDUA current signal measured by the miniEC and Epsilon electrochemical workstation at 1 s intervals for a potential of 400 mV applied for a duration of 60 s. The signals were in response to a 10μL droplet of 0 to 100 μM ferri/ferrohexacyanide placed directly on the IDUA.

Table 4.2: Summary of static electrochemical experiments showing steady-state signal, initial slope ((difference in signal)/10 s), and time needed to reach steady state for the miniEC and Epsilon systems.

Concentration of ferro- and ferrihexacyanide [μM]	Signal at 60s [nA]		Initial slope [nA/s]		Time to steady state [s]	
	miniEC	Epsilon	miniEC	Epsilon	miniEC	Epsilon
0	1.8 ± 0.5	2.0 ± 0.5	-5.2	-0.55	15	15
0.01	4.5 ± 0.5	2.9 ± 0.2	-3.9	-0.38	15	15
0.1	13.5 ± 0.9	10.6 ± 2.0	-2.9	-0.95	15	10
1	25.3 ± 1.2	15.6 ± 1.0	-2.23	-0.84	15	10
10	231 ± 4.8	180 ± 14.3	10.1	-1.84	12	6
50	1062 ± 20.7	1109 ± 122.9	58.2	-2.43	8	1
75	1551 ± 82.0	1820 ± 98.5	50.1	-3.51	5	1
100	2085 ± 132.64	2327 ± 40.4	67.5	-4.79	5	1



(a)



(b)

Figure 4.5: Comparison of signal-to-noise ratios of the steady state signals for the miniEC and BAS epsilon electrochemical systems in static droplet tests. The signal was taken at the 60 sec mark for combined potassium ferri- and ferrohexacyanide concentrations ranging from (a) 0 to 1 μM and (b) 10 to 100 μM.

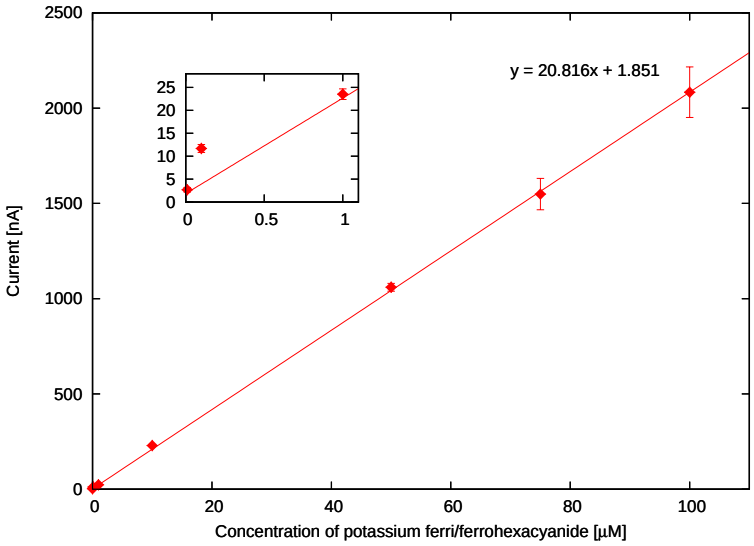


Figure 4.6: Dose response curve (triplicate analyses) for ferro/ferrihexacyanide with the miniEC plotting the steady state signals. A potential of 400mV was applied for a duration of 60 s. The signals were in response to a 10 μL droplet of varying concentrations of potassium ferri/ferrohexacyanide placed directly on the IDUA The background signal of buffer solution of 1.8 nA + 0.5 nA was subtracted from all data points.

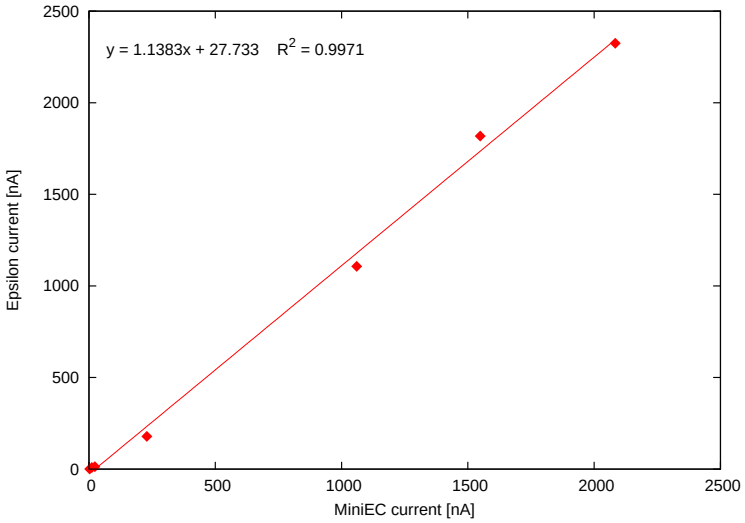


Figure 4.7: Correlation plot comparing the average of three miniEC measurements to the average of three Epsilon measurements of steady state IDUA current at 60s for combined potassium ferri- and ferrohexacyanide concentrations ranging from 0.01 to 100 μM. Background signals of 1.8 and 2.0 nA respectively, were subtracted from all data points. A high linear correlation of 0.997 was obtained.

was obtained. Thus, the miniEC proved to have a detection limit 10x lower than that of the standard lab-bench system. This sensitivity is most likely due to a number of factors. First, signal analysis of the miniEC electronic design ensured that components chosen were optimized for small signal detection. Furthermore, since signals routed within components or traveling shorter lengths are less likely to pickup noise, the smaller size and the integrated nature of the microelectronic chips used in the miniEC circuit helped to reduce the background noise in relation to the signal strength.

Comparing the dose response curves of the miniEC and the Epsilon set up (Figure 4.7) a linear regression analysis of the association resulted in the high correlation coefficient of 0.997. In addition, signal intensities obtained were very similar. This proves that the miniEC is a fully functional potentiostat for amperometric detection, and highly desirable due to its small size and low cost.

4.3.2 Dynamic electrochemical experiments

Proving to be an adequate amperometric system, the miniEC was further investigated using dynamic measurements in order to evaluate the response time behavior of the minipotentiostat. The microfluidic system was used as a flow injection device in which the sample (ferro/ferrihexacyanide) was injected into a continuous buffer stream. The microchannels had a constant flow of 0.1 M potassium phosphate buffer (1 μL/min) through one inlet. At approximately 60s intervals, a defined volume (20 nL, 100 nL, and 200 nL) of 10 and 100 μM potassium ferri/ferrohexacyanide was injected through the second inlet. Thus, typical temporal current peaks of a chromatogram were expected in this set up. The same analysis was carried out for the miniEC and the Epsilon set up.

The data of injections with 10 and 100 μM ferro/ferrihexacyanide solutions are

Table 4.3: Comparison of results for the amperometric microfluidic biosensor response to injections of 50, 100 and 200 nL of 10 and 100M potassium ferri/ferrohexacyanide using the miniEC and BAS Epsilon electrochemical workstation.

Volume of ferro- and ferri-hexacyanide injected [nL]	Average peak signal [nA]		Average area under peak [nA*s]		Signal-to-noise ratio of peak area	
	miniEC	Epsilon	miniEC	Epsilon	miniEC	Epsilon
50 (10 μM)	2.67 ± 0.28	2.10 ± 0.04	26.3 ± 0.25	28.4 ± 0.28	3.4	2.3
100 (10 μM)	5.17 ± 0.28	3.83 ± 0.15	59.00 ± 3.5	47.8 ± 1.5	7.6	4
200 (10 μM)	7.67 ± 0.28	5.27 ± 0.34	88.8 ± 2.8	66.5 ± 0.4	11.5	5.5
100 (100 μM)	28.2 ± 1.04	31.9 ± 0.70	339 ± 22.5	329 ± 8.5	5.4	9.1
200 (100 μM)	54.8 ± 0.56	63.6 ± 2.28	615 ± 11.3	620 ± 23.8	9.8	17.1

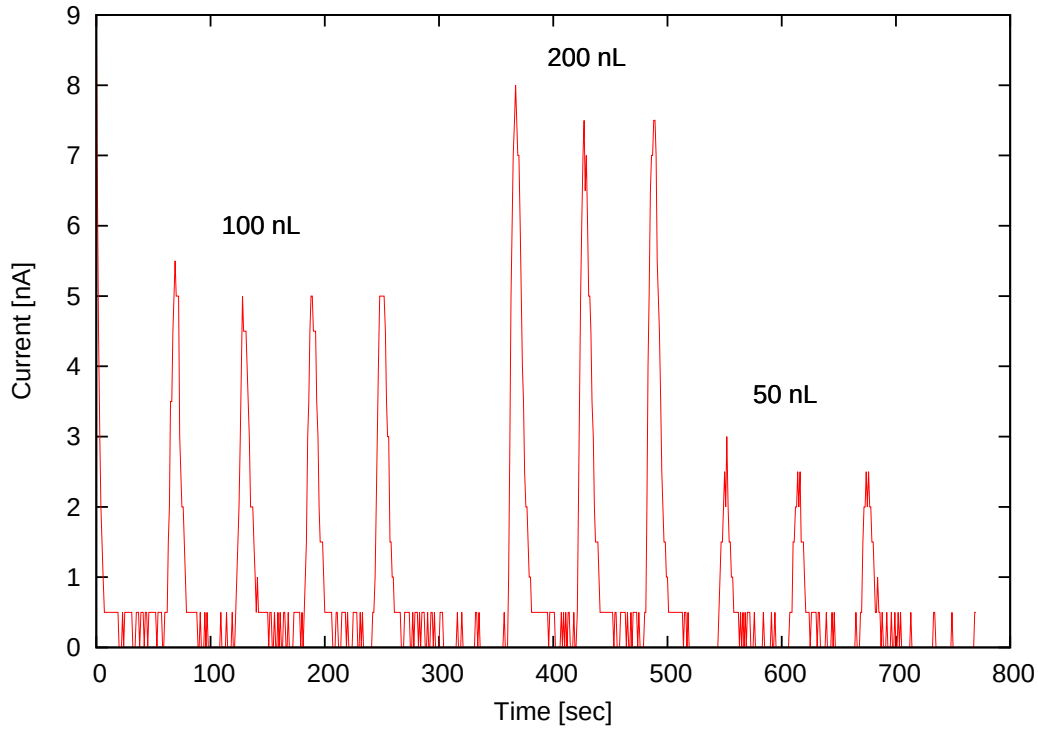


Figure 4.8: Profile of the sensor current signal measured by the miniEC in response to 3 injections each of 100, 200, and 50nL of 10 μM potassium ferri/ferrohexacyanide injected at a rate of 10 μL/min into a stream of buffer solution flowing across the sensor at a rate of 1 μL/min. Injections were done at approximately 60-second intervals.

summarized in Table 4.3. Sharp and well defined reproducible peaks were obtained for the miniEC system as shown by example for the 10 μM concentration in Figure 4.8, as also for the Epsilon set up (data not graphed). Comparing peak current signal and peak area, both electrochemical systems had very similar responses (Table 4.3). In general, the miniEC provided higher signals and higher signal to noise ratios for the lower concentration (between 48 and 109% higher signal to noise ratios) and lower values than the Epsilon for the higher concentration. This reflected the results obtained during the static experimental set up.

Comparing the concentrations obtained in this flow-injection approach, it was concluded from the peak current measured that the ferri/ferrohexacyanide was di-

luted approximately 10-fold by the buffer in the microfluidic flow-injection system. Since both electrochemical setups had limits of detections much below 1 μM this dilution caused no problem during this dynamic analysis. Table 4.3 also shows that both the peak height and the area under the peak signal (coulometric detection) were valid measures of the system response.

4.3.3 Detection of specific RNA sequences

The principle of detection of single stranded DNA and RNA molecules in the microfludic biosensor is based on a sandwich hybridization reaction of two oligonucleotides, capture and reporter probes, with the target molecule. The reporter probes were tagged with liposomes encapsulating potassium ferri- and ferrohexacyanide marker molecules. The capture probes were immobilized on superparamagnetic beads. The probes, target sequence and a hybridization buffer were mixed, incubated for 10 minutes and subsequently applied to the microfluidic biosensor via inlet 1. The resulting sandwich complexes were captured on the magnet directly before the electrode located in the microfluidic biosensor cartridge (Figure 4.2). Subsequently, liposomes were lysed using a detergent releasing the ferri- and ferrohexacyanide flowing past the IDUA for quantification. The miniEC maintained a potential difference of 400 mV across the IDUA and the current monitored was directly proportional to the amount of ferro/ferrihexacyanide, which in turn was directly proportional to the number of liposomes lysed and thus to the amount of target sequence caught on the magnetic beads in the capture zone.

Figure 4.9 shows typical current peaks measured by the miniEC in the presence and absence of Dengue virus RNA. The positive peaks had an average retention time of 60 2 s and had peak values that ranged from 7 to 10 times higher than the negative controls under current experimental conditions. In Figure 10, for

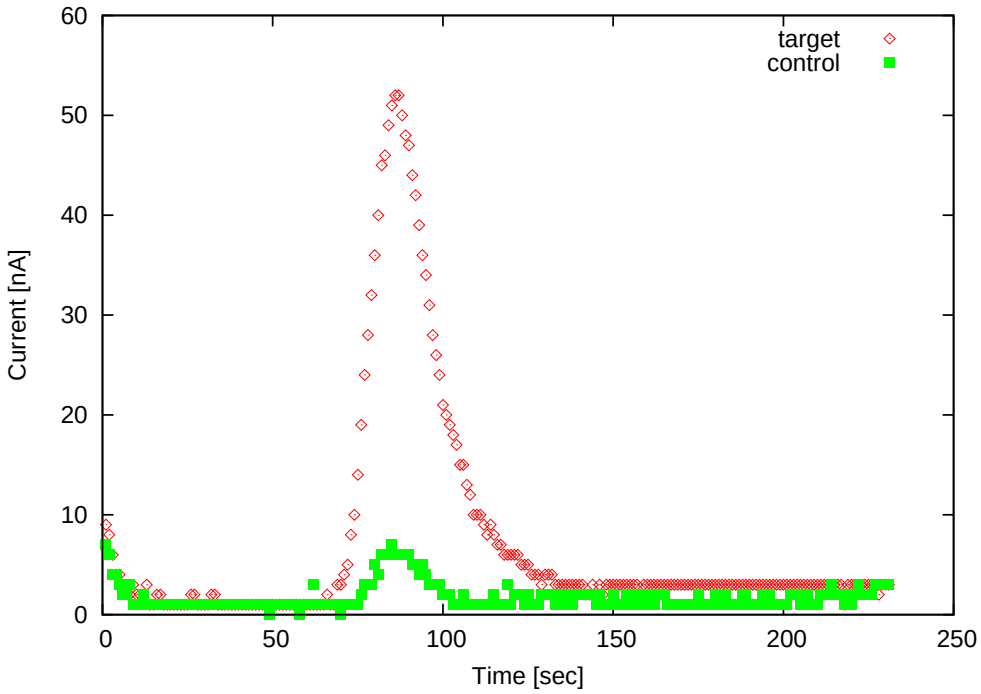


Figure 4.9: Current peaks detected by the miniEC in the presence (target) and absence (control) of target RNA in sample solutions flowing through the biosensor cartridge.

example, the peak signal in the absence of target RNA was 3 nA versus 26 nA when target RNA is present. The presence or absence of target RNA was thus apparent within 2 minutes of sample injection.

4.4 Conclusion

The integration of a minipotentiostat, an interdigitated ultramicroelectrode array and a microfluidic biosensor for the detection of nucleic acid sequences has been successfully demonstrated. A small, fully portable and inexpensive detection system (less than $50) has thus been integrated with a microfluidic channel system that is considered to be an interesting alternative to complex optical detection systems currently used in most diagnostic, genomic and proteomic analyses. The miniEC is capable of quantifying electrochemically active components in the

microchannel system. While in this study only constant potential amperometry has been used, the microcontroller-based design of the miniEC instrumentation allows it to be reprogrammed as needed in the future to perform other types of amperometric analyses.

The PG580 (Uniscan Instruments, U.K) and PalmSens (Palm Instruments, Netherlands) are examples of handheld potentiostats about the size of a PDA that are commercially available for electrochemical analysis and cost >$1000 each. The configuration of the miniEC system however lends itself well to the mass production of inexpensive integrated single use disposable diagnostics with applications in medical diagnostic, research and biosecurity analyses, similar to the highly successful credit-card sized glucose sensors available in most drug stores for about $50 each. Most of these glucose sensors are also based on amperometric detection measuring the oxidation or reduction current of electrochemical mediators such as ferrocene on screen printed electrodes. These measurements require 3 to 10 μL of blood and usually take a few minutes.

CHAPTER 5

SCALABLE FIRMWARE DESIGN AND IMPLEMENTATION

5.1 Analysis

The miniEC application firmware directs the operation of the msp430 microcontroller (MCU) and provides an interface through which the user can control the system. In terms of operation, the MCU must power the IDUA biosensor and read analog amplified sensor signals at timed intervals. The sensor reading must then be digitally filtered, formatted and transmitted to a PC or stored in a time-stamped file in the MCU's memory. To achieve this capability, the MCU firmware must program

- timers for the interval timing and a real-time clock;
- the ADC peripheral for reading the IDUA signal;
- the DAC peripheral for providing the IDUA excitation potential;
- the USART peripheral for serial communication;
- flash memory for data storage;
- the LCD controller for data display;
- and its I/O ports to receive signals and send signals to external peripherals.

In addition, these programmed activities must run only in response to environmental events. The miniEC is connected to its environment through sensors, actuators and indicators. The sensors provide the MCU with information about the current state of the environment and the MCU communicates its response through the actuators and indicators.

The miniEC‘s environment is composed of actors it interacts with but has no control over. The context diagram in Figure 5.1 identifies four actors: a user,

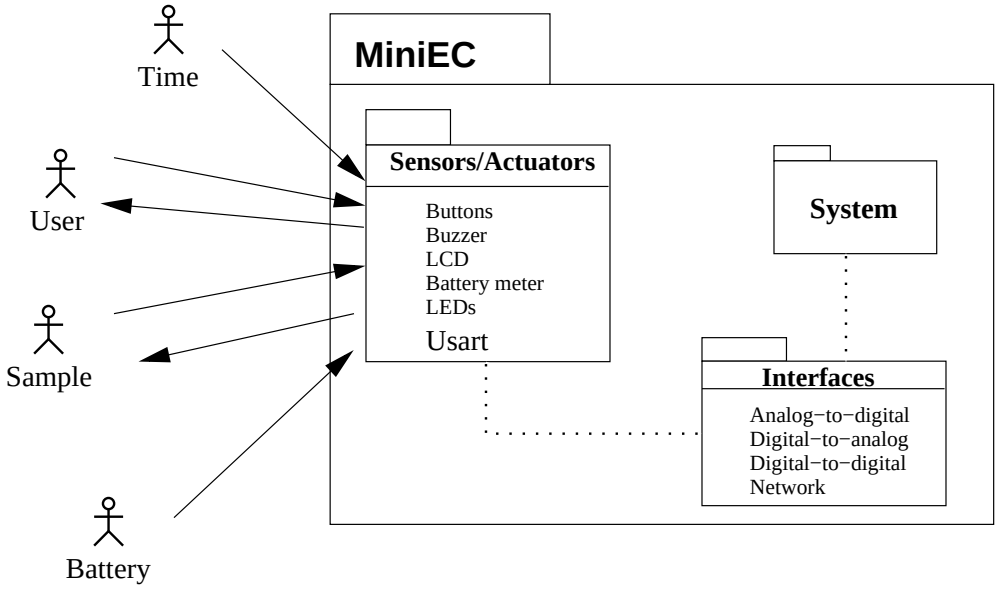


Figure 5.1: MiniEC context level diagram illustrating the interaction between a miniEC and the user, time, battery and sample as agents that can initiate activity.

the sample to be analyzed, time and the battery that make up the miniEC's environment. It also shows the interfaces, the sensors and the actuators that allow the miniEC to exchange messages with the agents.

The sample agent‘s sensor is the IDUA while the battery agent‘s sensor is the low battery output of the voltage regulator in the power module. The passage of time is monitored by the microcontroller's internal timers clocked by a 32KHz crystal oscillator. The user agent's requests are sensed by four momentary push-buttons labeled “Stop”, “Enter”, “Up” and “Down”. These correspond to the buttons labeled B1 to B4 in the circuit schematic (Appendix A). A fifth push-button, (labeled RST) is used to reset the miniEC to default values should it malfunction during operation. The final user control mechanism is through a serial connection to a PC or PDA running the miniEC graphical user software. The miniEC in turn responds to its environment with the LED, buzzer and LCD or through the serial communication module to the PC.

In addition to responding correctly to external events, the miniEC is required to respond in bounded time. A correct response outside the expected time is considered suboptimal. An example would be failure to update time every second. Clearly this is undesirable as the stored data will have incorrect timestamps. This may or may not be important depending on the field in which the miniEC is being used. In a stand-alone long-term monitoring application for instance where pathogen levels may be transient, one wants to know exactly when contamination occurs. An incorrect timestamp on the data collected is a serious problem. It is however not a problem in one-shot measurements where the result is read immediately.

Further analysis of the miniEC's environment in view of the miniEC's functional requirements and hardware design identifies events that are triggered by agents or the miniEC itself. The events, the system response, and the upper bound of response times are summarized in Table 5.1. The events are also characterized by direction and type. Events originated by the miniEC itself have an "OUT" direction while events received by the miniEC have an "IN" direction. Finally, an event type can be either periodic if it occurs at fixed intervals or episodic, if it occurs randomly.

From Table 5.1, it is clear that some events elicit different responses depending on the status of the system. Pressing the 'enter button for instance may lock in a selection from a list of options or start recording data from the biosensor depending on the context. Thus the next step in the miniEC rmware design was the development of a nite state machine (FSM) model that describes the way the system changes based upon external events, and what actions it takes as a result of those state transitions.

Table 5.1: Events triggered during miniEC operation and the system response expected.

Event	**MiniEC Response**	**Dir**	**Type**	**Time**
User presses stop button	Stop current task	In	episodic	1 s
User presses enter button	Lock in a menu selection or start a task	In	episodic	1 s
User presses up button	Move up a menu or increment a selected value	In	episodic	1 s
User presses down button	Move down a menu or decrement a selected value	In	episodic	1 s
User presses reset button	Reload default settings	In	episodic	1 s
Low battery alarm	Warn user and stop current tasks	In	episodic	1 s
A second passes	Update real time clock	In	periodic	0.5 s
A sample interval passes	Take a reading, display and log or transmit reading	In	periodic	1/2 interval
A sample duration passes	Log or transmit data trailer	In	periodic	1 s
Enter idle mode	Switch off peripherals and enter low power mode	Out	episodic	1s
Write to USART	Transmit information to PC	Out	episodic	baud rate
Read from USART	Retrieve and interpret information from PC	In	episodic	baud rate
Memory is full	Warn the user	Out	episodic	1 s
Enter active mode	Leave low power mode, power up peripherals and process pending tasks	In	periodic	6 μs

A Finite State Machine (FSM) is a very powerful way of modeling and implementing the control logic for applications. A state in the state model provides context for the actions that the system exhibits at any moment in its operation. Events of course are inputs to a state while an action is simply a set of instructions to be executed. States are connected by transitions. A transition defines conditions that need to be met before the system moves from one state to another. An FSM must have a state to start it off. It then keeps track of the current state

and operates on system inputs that trigger transitions to new states. There are several advantages to using the state machine approach. It allows one to break the behavior of a complex system into states and events with well defined conditions for state transitions. Also, given a set of inputs and a known current state, the state transition can be predicted, allowing for easy system simulation and testing. The technique also encourages modular design and flexibility. Furthermore, on the hardware side, since only the code for the current state is executed at a time, there is low processor overhead on the microcontroller. Another advantage to using the state machine model, is that the system can be visualized with meaningful state graphs that can be readily transferred to the code implementation in either assembly language or a higher level computer language.

5.2 The MiniEC Finite State Machine

The miniEC's FSM states are derived from the functions that exhibit state behavior from the perspective of a user doing an electrochemical analysis. Five top-level states were identified. Figure 5.2 shows the relationship between the states and the events that cause the miniEC to transition from one state to another. The user operates the device by pressing one of the 4 buttons or by sending commands mimicking button events via the serial connection.

In the FSM, the miniEC starts out doing nothing more than keeping time with all peripherals but the basic timer and LCD turned off. The LCD displays the time. This is the IDLE state. When the user loads a sample and presses the enter button, the miniEC enters a READY state where all the peripherals necessary for running a test and recording data are turned on if they are off and initialized. Memory is checked for sufficient space to store the test data for the duration specified by the user. If there is insufficient memory, an alarm is posted

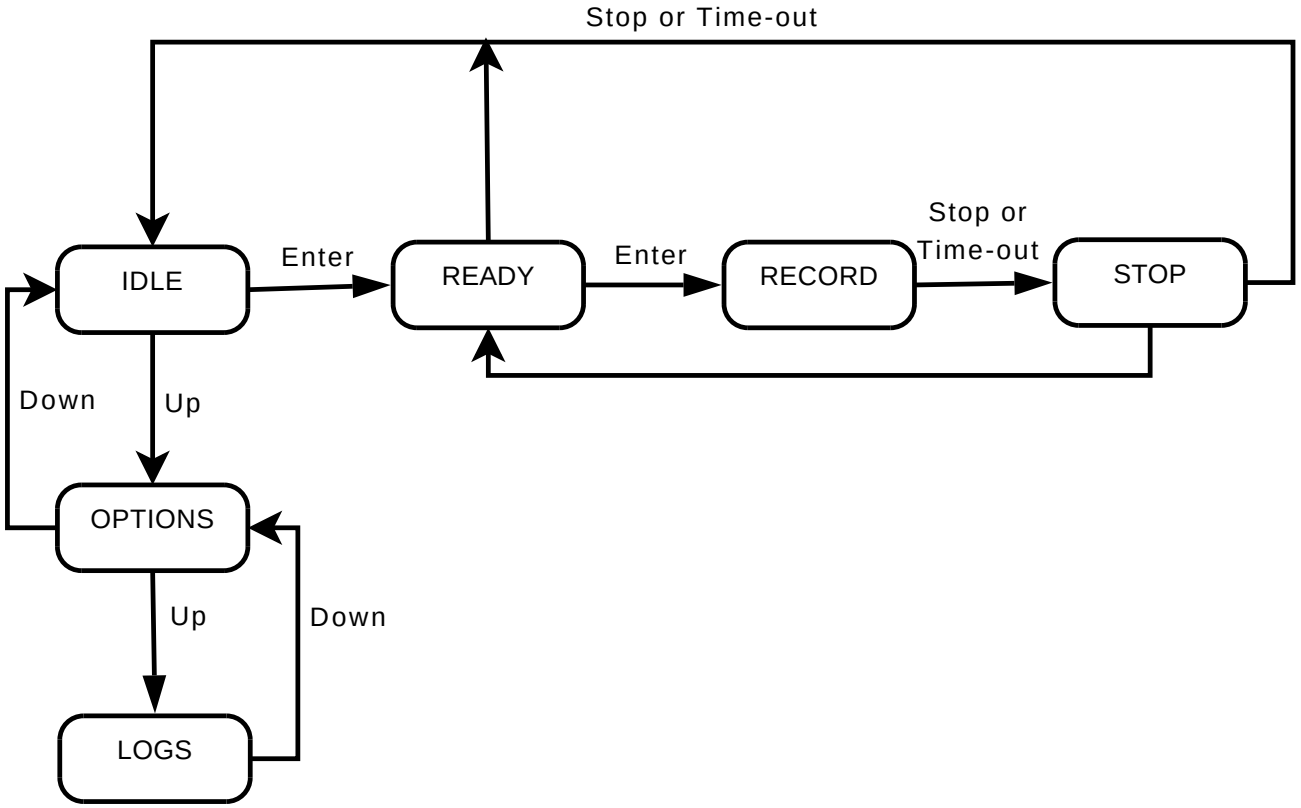


Figure 5.2: MiniEC state chart. The device starts out in the idle state. When the enter button is pressed, the miniEC transitions to the ready state. The user may revert back to the idle state by pressing the stop button. In the ready state another enter button event transitions the miniEC into record state. During recording, the test can be stopped by pressing the stop button. In stop mode, pressing the enter button will allow the test to re-enter the ready mode, however if the stop button is pressed a second time, the test is completely stopped and the device re-enters the idle mode. The stop and ready modes have timeout periods. If no events occur within the timeout periods, the miniEC will return to the idle mode.

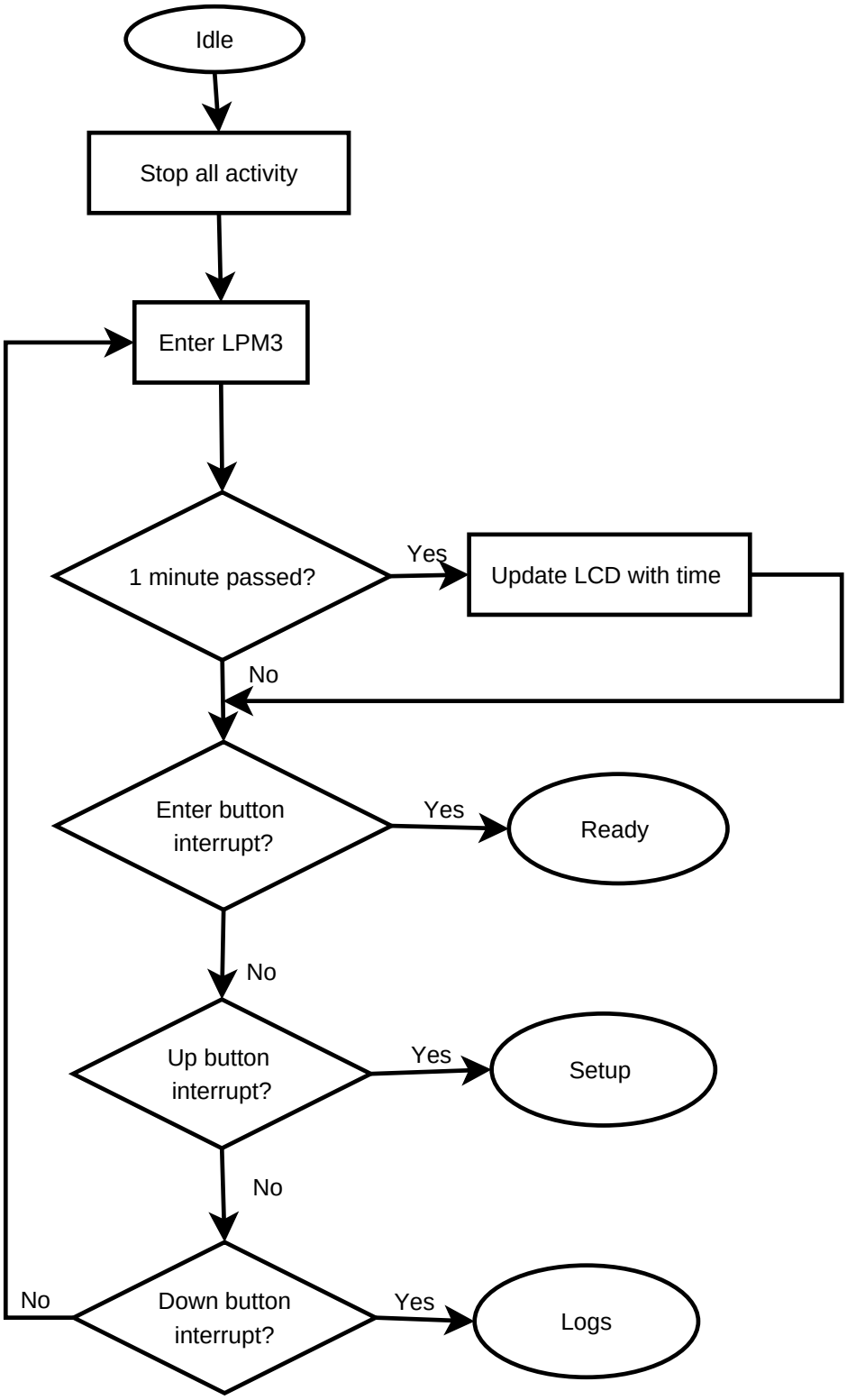


Figure 5.3: Idle activity flowchart

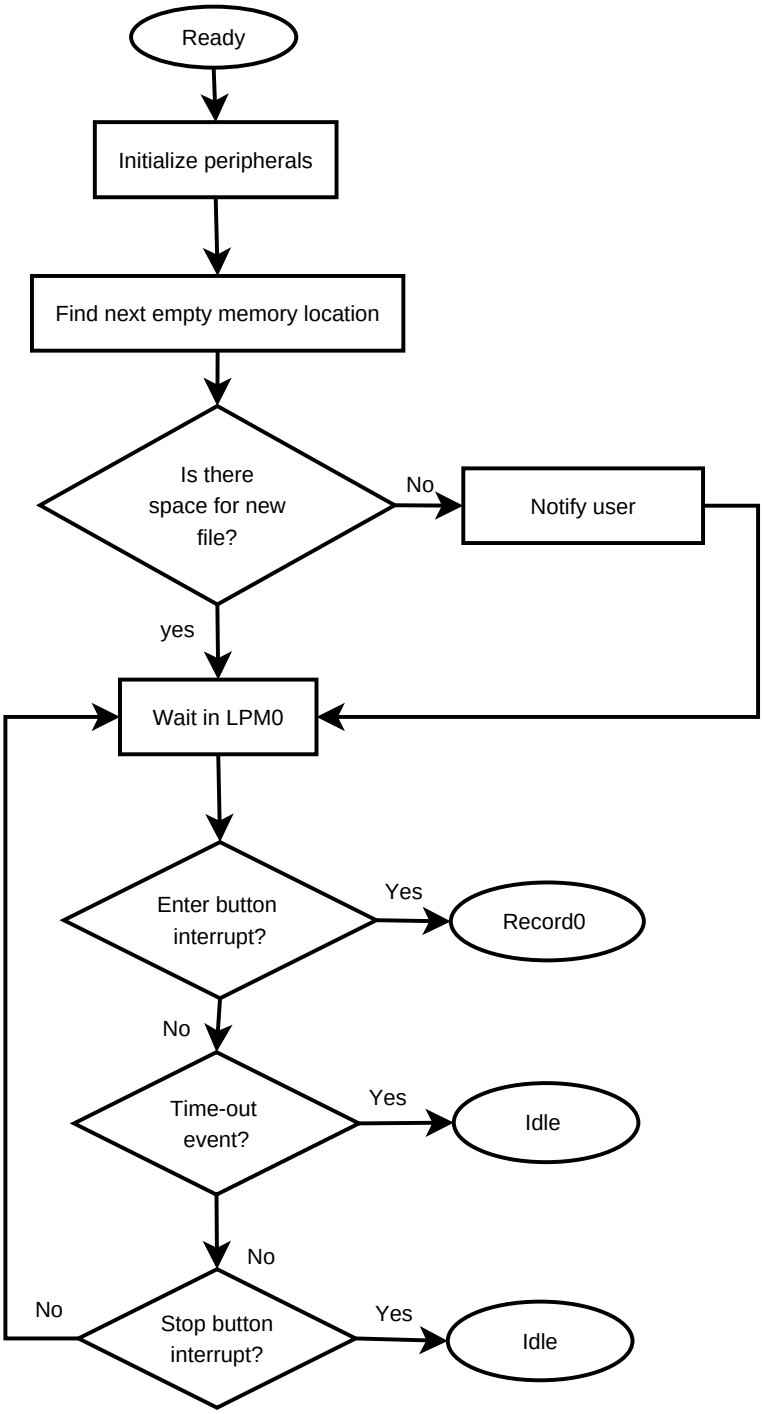


Figure 5.4: Ready activity flowchart. Peripherals needed for the test run are initialized first. After a check for memory to store the test data, the system waits for the user to run or abort the experiment by pressing the enter or stop button respectively. If there is no user input for a set interval, the system reverts to the IDLE state.

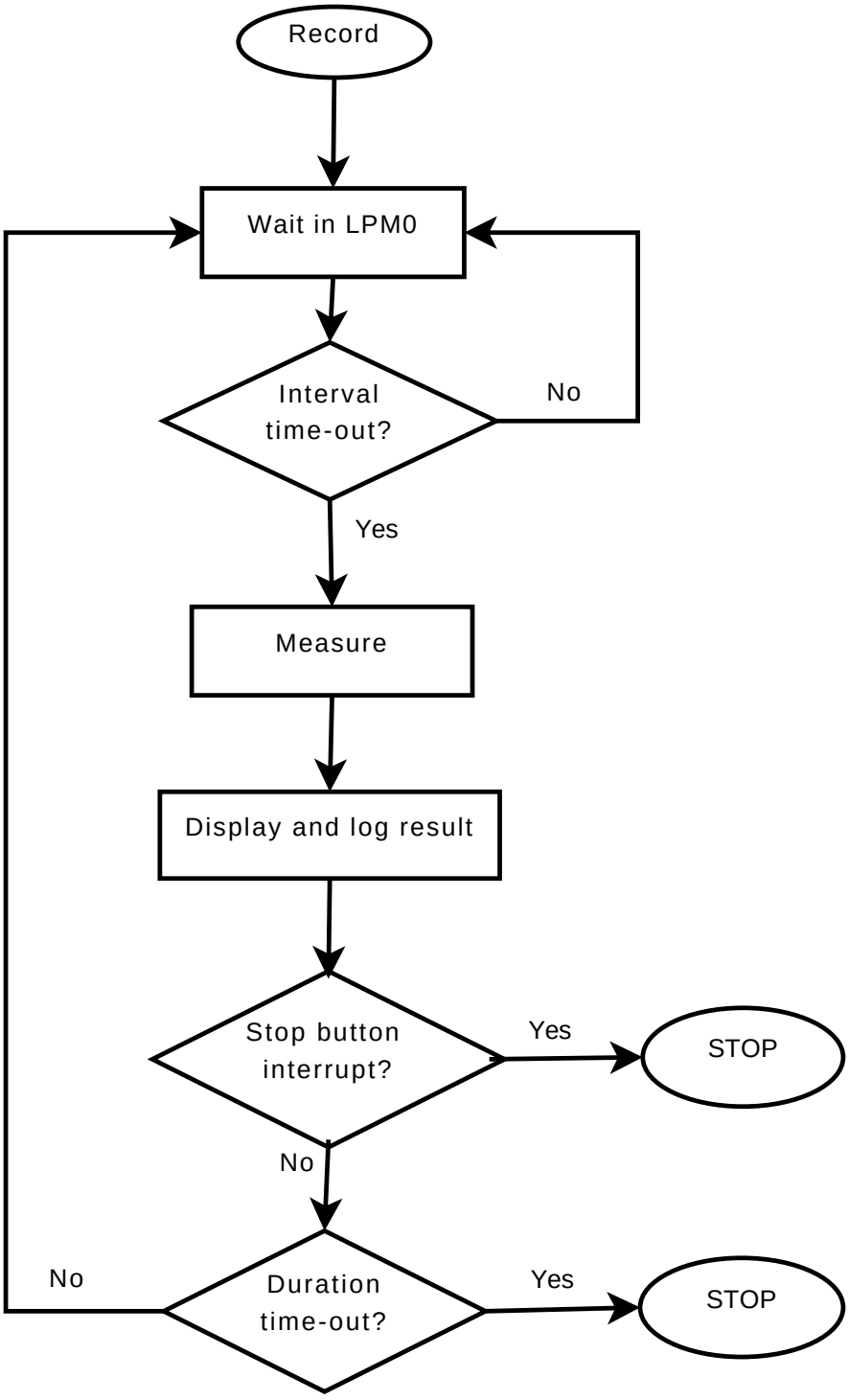


Figure 5.5: Record activity flowchart. For the duration specified by the user, a measurement is made, stored and displayed at every interval unless the user presses the stop button. At the end of the duration or stop event, the file is closed and the system enters the STOP state.

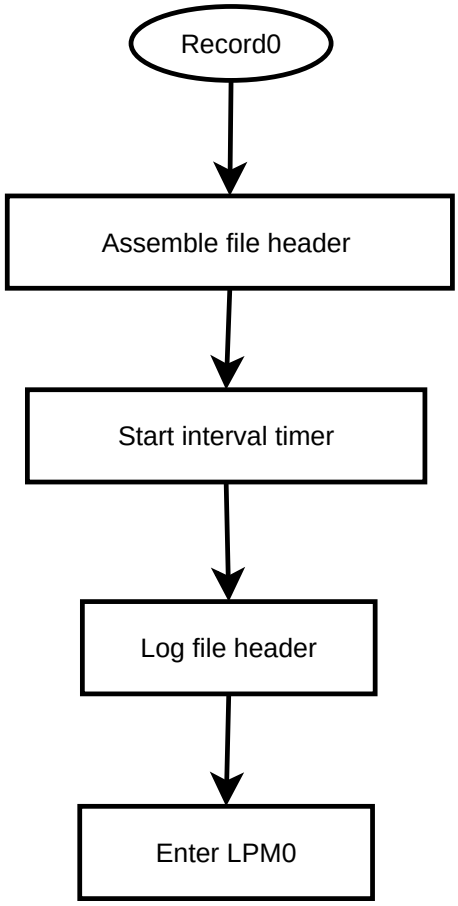


Figure 5.6: First record (record0) activity flowchart. This refers to actions taken when the system first enters RECORD state from the READY state. Here, the interval timer is started and a file opened for subsequent measurements.

on the LCD display otherwise the display is cleared and a file is opened to receive data. Pressing the enter button again puts the miniEC into the RECORD state. A potential is applied to the IDUA. At the preset intervals, its output is read, processed, displayed on the LCD and stored or transmitted to a PC. An alarm for insufficient memory in the READY state will not prevent the user from proceeding into the RECORD state. Everything will proceed as normal but data storage will simply stop when the system runs out of memory. At the end of the test duration or if the user presses the stop button during a test, the system enters the STOP state. In this state, The LCD displays the last data point measured and the test

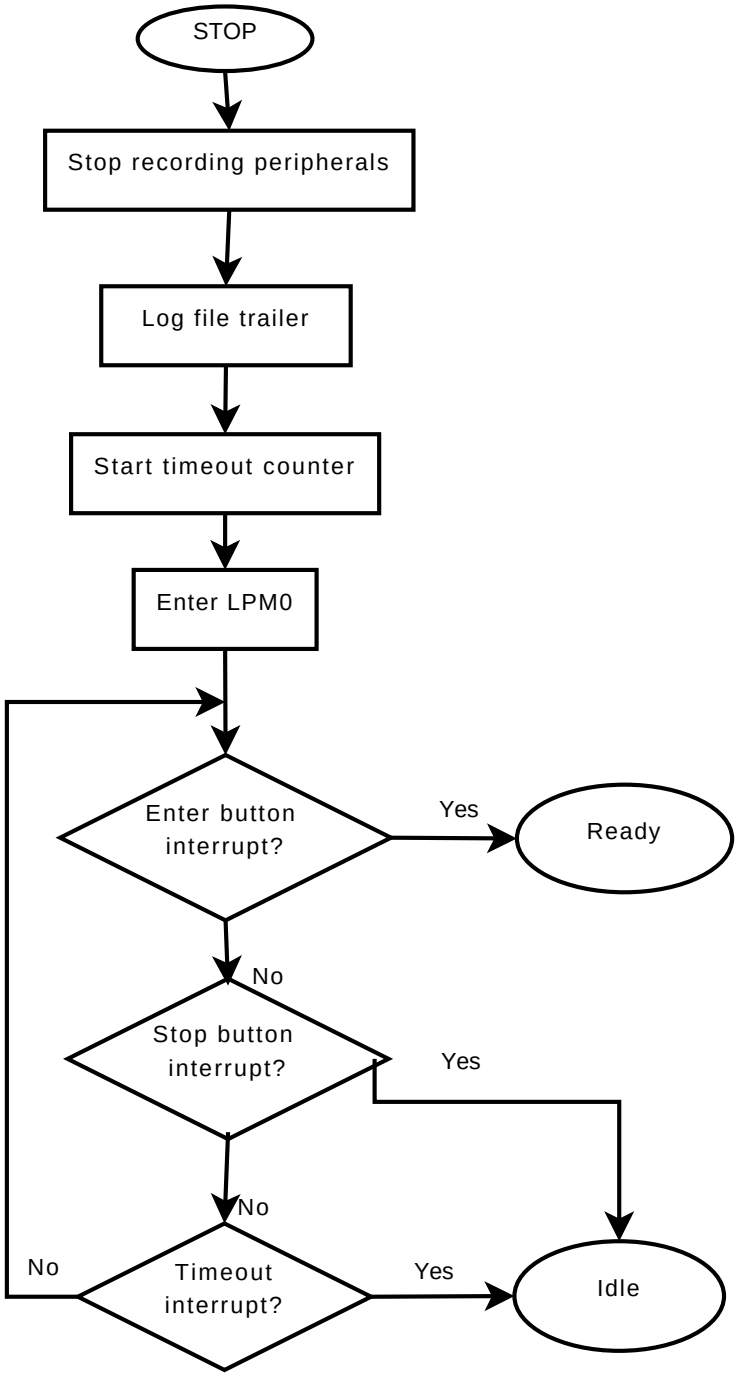


Figure 5.7: STOP state functional flowchart. The interval timer is stopped and the system waits in low power mode 0 until the user presses either the enter button to prepare for another experiment or the stop button to enter the IDLE mode. If there is no user input for a set interval, the system reverts to the IDLE state.

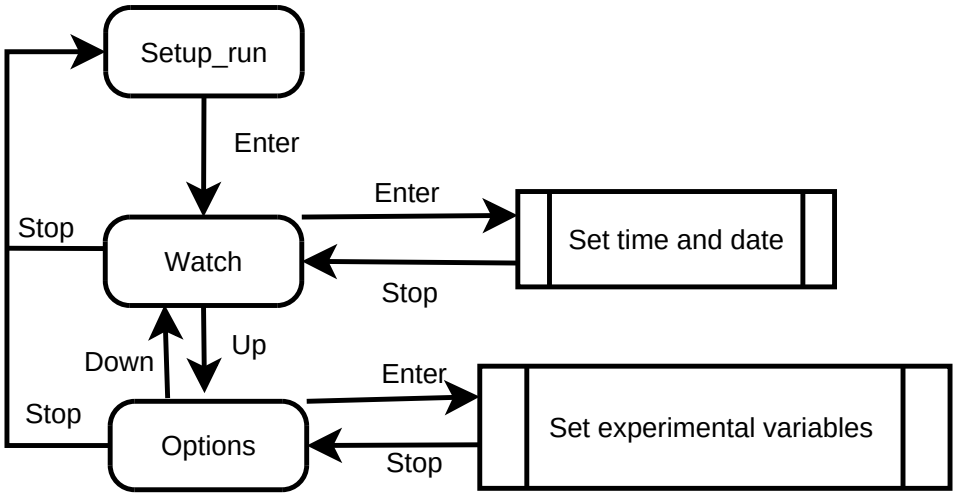


Figure 5.8: Setup state chart

file is closed to further input. All the necessary peripherals remain on so another test can be run again quickly. The user can exit the STOP state by pressing either the stop or enter button. Pressing the stop button puts the system back into the IDLE state. Pressing the enter button puts the system into the READY state. The current implementation of a STOP instead of a PAUSE state has to do with how data is stored by the miniEC. Data is conserved by not recording the time at which every data point is collected. Instead, given the constant interval between measurements, just the start time of the test is recorded and the time at which a measurement occurs is inferred from the start time and the position of the data within the file. The behavior of the STOP state can easily be converted to a PAUSE state that resumes a suspended test run instead of opening another file. This requires that the logging function also be modified to note the period of inactivity.

The STOP and READY states have timeout intervals associated with them. At the end of the timeout period, the system reverts to the IDLE state. From the IDLE state, two other states named SETUP and LOGS can be reached by pressing the up and down buttons. In the SETUP state the user can change the miniEC

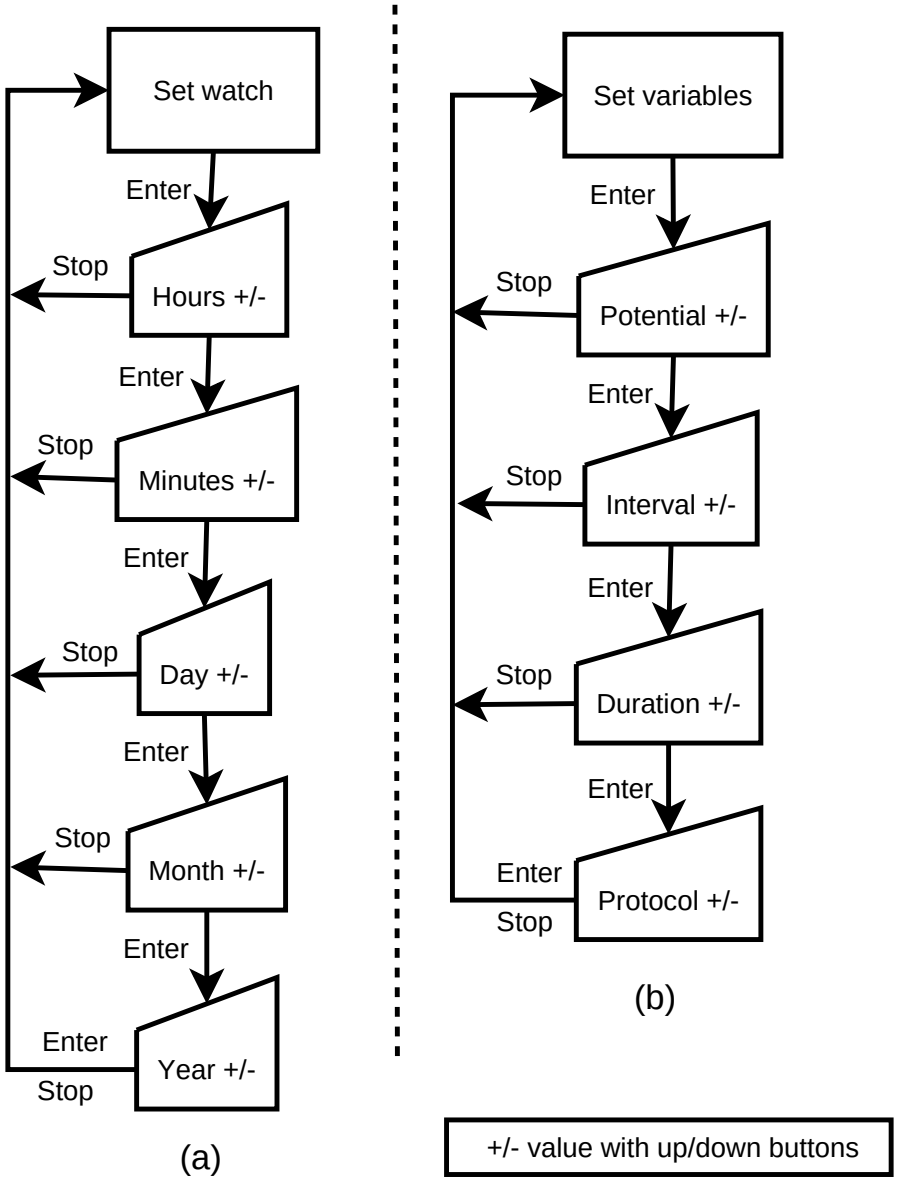


Figure 5.9: Within setup the user may select the watch or settings options to update the time and experimental variables respectively. For each variable, the user may increment or decrement a value with the up/down buttons. Pressing the enter button moves the menu forward. Pressing the stop button takes the user to the top of the setup menu.

settings such as the time, date and configure the test parameters such as the time interval between sensor readings, duration of the test and the excitation potential driving the sensor reactions. In the LOGS state the user can scroll through the recorded data files and view summary statistics such as peak and average signal.

It must be noted that, the miniEC can be run either as a standalone device through the use of its button interface or under PC control. An event of interest that is not shown in the state chart in Figure 2 is a command from the serial interface that puts the device under PC control. The FSM will run the same way with either control interface. The graphical user interface on the PC has buttons and other controls for starting and stopping experiments, configuring miniEC settings and viewing data just like the push-button interface. The only difference is in the RECORD state. Under PC control, the data is transmitted to the PC instead of being stored in the miniEC's memory. Like the real time clock, the PC interrupt runs in the background outside the FSM loop so the miniEC can respond to a PC command in any state.

Figures 5.3 to 5.9 show the activities that occur in the various states mainly in response to the buttons being pressed or timeout intervals. In the SETUP state where a variety of decisions have to be made about experimental variables, the decision logic is implemented as local FSMs. In the LOGS state, using the up and down buttons will cycle through stored data files showing the peak and average value measured for each file.

Since the states or modes of operation are for the most part independent environments, a new mode can be inserted anywhere in the main or sub state machines without changing the user interface. Software development and subsequent upgrades are thus simplified.

5.3 Implementation

To ensure portability of firmware to new MSP430 MCUs or a different MCU family, the embedded software was written in ANSI C using the free mspgcc tool chain. The firmware is divided into three categories: the main module, FSM module, utility modules. The main module, gives the overall flow of the entire firmware logic. The function of the FSM module was discussed in previous sections.

The utility modules implement an interface for the LCD, hardware peripherals, file storage and real time clock. This way if a different LCD is used, only the code in the LCD module needs to be changed. Likewise, if a different microcontroller is used, only code for the particular peripheral that is different between microcontrollers needs to be changed in the hardware module. In the current miniEC implementation, the microcontroller supplies the timer to implement the real time clock. Many applications however prefer to use an external hardware component dedicated to this function. Separating the real time clock code into its own module allows a developer to add and code for this function without impacting the rest of the firmware.

In addition to the noted modules, a single header file (globals.h) was defined to contain all the variables and constants that are common to all the modules. This way each variable and constant can be unambiguously defined and updated in one place. The first iteration of the firmware did not have the structure discussed here abstracted into separate files, instead the functions were grouped in the various categories within one file. Only the lcd code was in a separate file. The structure was imposed to manage the growing complexity of the code as the system was developed from a single sensor into a multi-sensor device. The complete listing for the first iteration and the subsequent modularized implementation is given in sections A.1 and A.2 of the Appendix.

5.3.1 The main module

The firmware logic is event driven. The MCU is programmed to run in low power mode 3 (LPM3 or idle mode) until an interrupt wakes up the CPU. The MCU returns to the LPM3 as soon as it completes the action associated with the event (Figure 5.10).

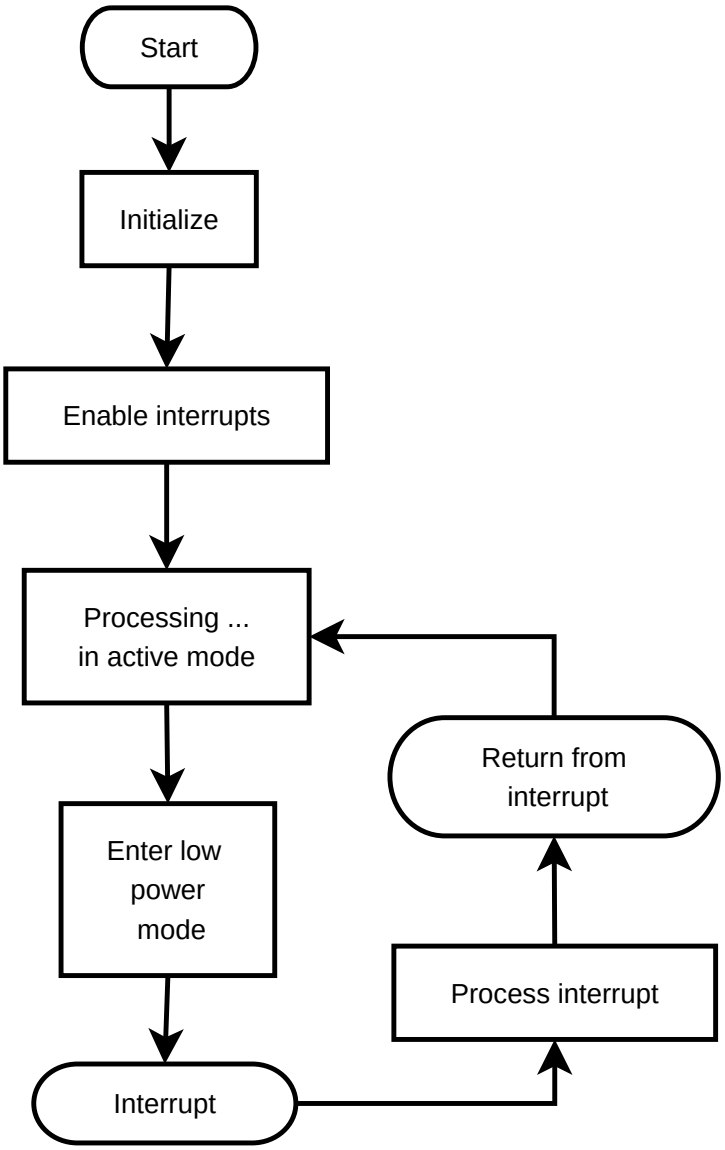


Figure 5.10: Microcontroller firmware logic flow. Operation is interrupt driven. The microcontroller stays idle until an interrupt occurs. It then enters active mode and performs the action requested by the interrupt

On startup, the microcontroller is programmed to initialize program variables such as the location of data memory and time/date to default values. Various peripherals are also initialized starting with the watchdog timer and clock: The watchdog timer is disabled and the MCUs master clock is programmed to run at 1, 4 or 8 MHz. The basic timer which serves as the pulse for a real time clock is setup to cause an interrupt every second. LCD and the USART peripherals for serial communication are enabled as well. The rest of the pins are initialized for their I/O functions. Unused I/O pins are configured as outputs. After this setup, all interrupts are enabled. The microcontroller then proceeds into a main routine loop that logically implements the miniEC's finite state machine. In the main loop, the system mostly resides in low power mode 3 (LPM3) where the master clock and CPU are disabled. Enabled peripherals remain active. The basic timer and USART for instance remain active by running off of the 32 KHz auxiliary clock. When an event occurs, the event interrupt automatically switches on the internal fast master clock and restores the CPU which executes the instructions necessary to respond to the event.

5.3.2 The FSM module

Finite state machines in C, are often created by using multiple switch statements within a main switch statement. The main switch statement takes the current state as an input and within each states switch statement, one will need to implement another set of switch statements for each possible event.

While this implementation is acceptable for very small state machines, it is not easy to debug or scale and uses RAM inefficiently. Since the MSP430 microcontrollers architecture and instruction set are optimized for fast table lookups, the implementation we chose uses a lookup table algorithm that translates the state

chart of the miniEC FSM in Figure 5.2 into a state-event table (Table 5.2) that shows event transitions.

The table is rendered in code as a 2-dimentional array with the current state and event used as keys to retrieve an entry. Each entry in the state-event table is a structure that contains a pointer to the action to be executed and the value of the next state. The steps to implement the FSM are summarized in Table 5.3. Table 5.4 shows how the FSM fits into the main module algorithm.The full listing can be found in Appendix B. The state-event table created in the implementation is stored in program memory so only 3 bytes of RAM for the value of the current state, previous state and event tray is required to run the FSM. It can also be scaled up or down easily by updating the states, events and state-event table.

5.3.3 The Utility modules

Hardware

The hardware module provides methods for initializing and accessing the ADC, DAC, and USART peripherals of the microcontroller. It also configures the ports for button, LED and buzzer operation.

Real time clock

The MSP430's basic timer is programmed by the firmware to implement the time agent that keeps the miniEC system aware of the passage of real time in the environment. Clocked by the MSP430's auxiliary clock (ACLK) at a speed of 32 KHz, the miniEC keeps time by maintaining a count of seconds, minutes, hours, day-of-the-week, day-of-the-month, and year. The actual mechanism of counting time is achieved with a basic timer overflow interrupt algorithm illustrated by the flowcharts in Figures 5.11 and 5.11. The basic timer is set to overflow every

Table 5.2: Main state-event transition table: Entries in the table indicates the action and the next state when the event indicated by the column heading occurs during the state indicated by the row heading.

State/Event	Stop	Enter	Up	Down	Duration timeout	Interval timeout
IDLE	none IDLE	ready READY	setup SETUP	logs LOGS	none IDLE	none IDLE
READY	idle IDLE	record0 RECORD	none READY	none READY	idle IDLE	idle IDLE
RECORD	idle STOP	none RECORD	none RECORD	none RECORD	stop STOP	record RECORD
STOP	idle IDLE	ready READY	none STOP	none STOP	idle IDLE	none STOP
SETUP	idle IDLE	setup_run SETUP	logs LOGS	idle IDLE	none SETUP	none SETUP
LOGS	idle IDLE	logs_run LOGS	idle IDLE	setup SETUP	none LOGS	none LOGS

Table 5.3: MiniEC finite state machine implementation

Require: Events
Require: States
Require: Actions
Require: Event tray

1: Define state-event table
2: Initialize event tray, current state and previous state
3: Feed events sourced by interrupts (eg. pressing a button) and events from polling (eg. from the serial communication buffer) into the event tray
4: Retrieve an event from the event tray in order of priority.
5: Execute the associated action
6: Perform the transition dictated by the state-event table.

Table 5.4: The MiniEC main function algorithm

```
1: initialize variables and hardware
2: initialize FSM
3: enable interrupts
4: loop
5:    Update time if timer interrupt has occurred
6:    Check command buffer for serial command
7:    if if command found then
8:       update event tray with command
9:    else
10:      update event tray with interrupt sourced event
11:   end if
12:   Run FSM
13:   repeat
14:      get the current state
15:      get the next event from event buffer
16:      execute action
17:      set the previous state variable to current state
18:      transition into next state
19:   until event tray is empty
20:   sleep in LPM3 until next interrupt
21: end loop
```

second and generate an interrupt to wake up the MCU. In active mode, the MCU increments the timing variables and performs all calculations and checks required, such as for length of month, to ensure a correct calendar.

File storage

Flash memory differs in several ways from traditional disk memories. The most notable feature of flash memory is that it has no moving parts. This makes the flash chip shock resistant and well suited for portable applications where devices are likely to get dropped. Flash memory is also ideal for battery-powered systems since there is no need to spin up a hard drive. One limitation of flash memory is that while it can be read and programmed a byte or a word at a time it must be erased a segment at a time. Also once a byte has been programmed it cannot be reprogrammed unless the whole segment is erased first. Since the flash memory has a limited number of write-erase cycles applications that store frequently changing data often require a tailored flash file system to minimize wear.

The MSP430 MCU has 60 KB of in-application re-programmable flash memory partioned into two sections: 20 KB for firmware and 40 KB for data. The file storage module provides methods to read and write to the data partition. See the full listing in Appendix A.

The miniEC requirements stipulate storage for at least 100 basic tests. To determine how much memory this is, a simple file format was first defined in table 5.5 for data storage. It consists of a file header that lists the test parameters of potential, interval and duration preset by the user, and the time a test was started. This is followed by the actual test data recorded and a trailer, which records the maximum and average values.

Numerical markers instead of string markers were chosen to mark the start of the file as a whole and to mark the boundary between data and header/trailer

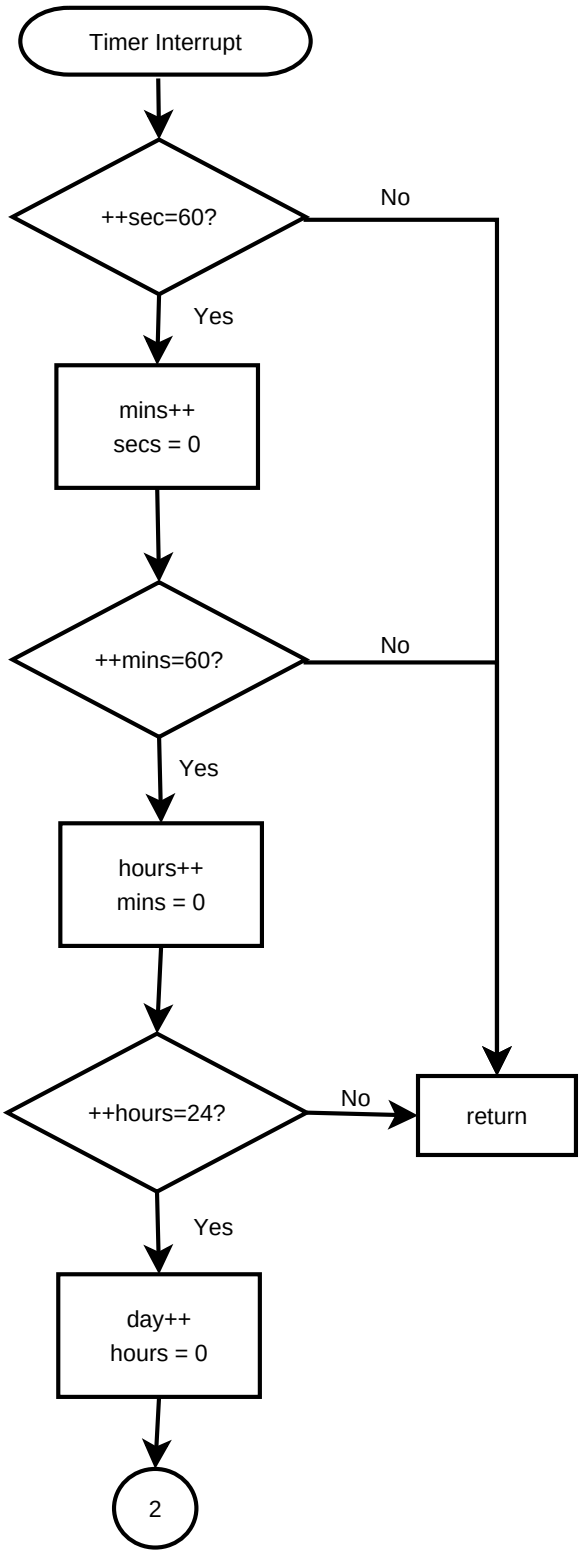


Figure 5.11: Flowchart illustrating how time is updated every second in the real time clock module.

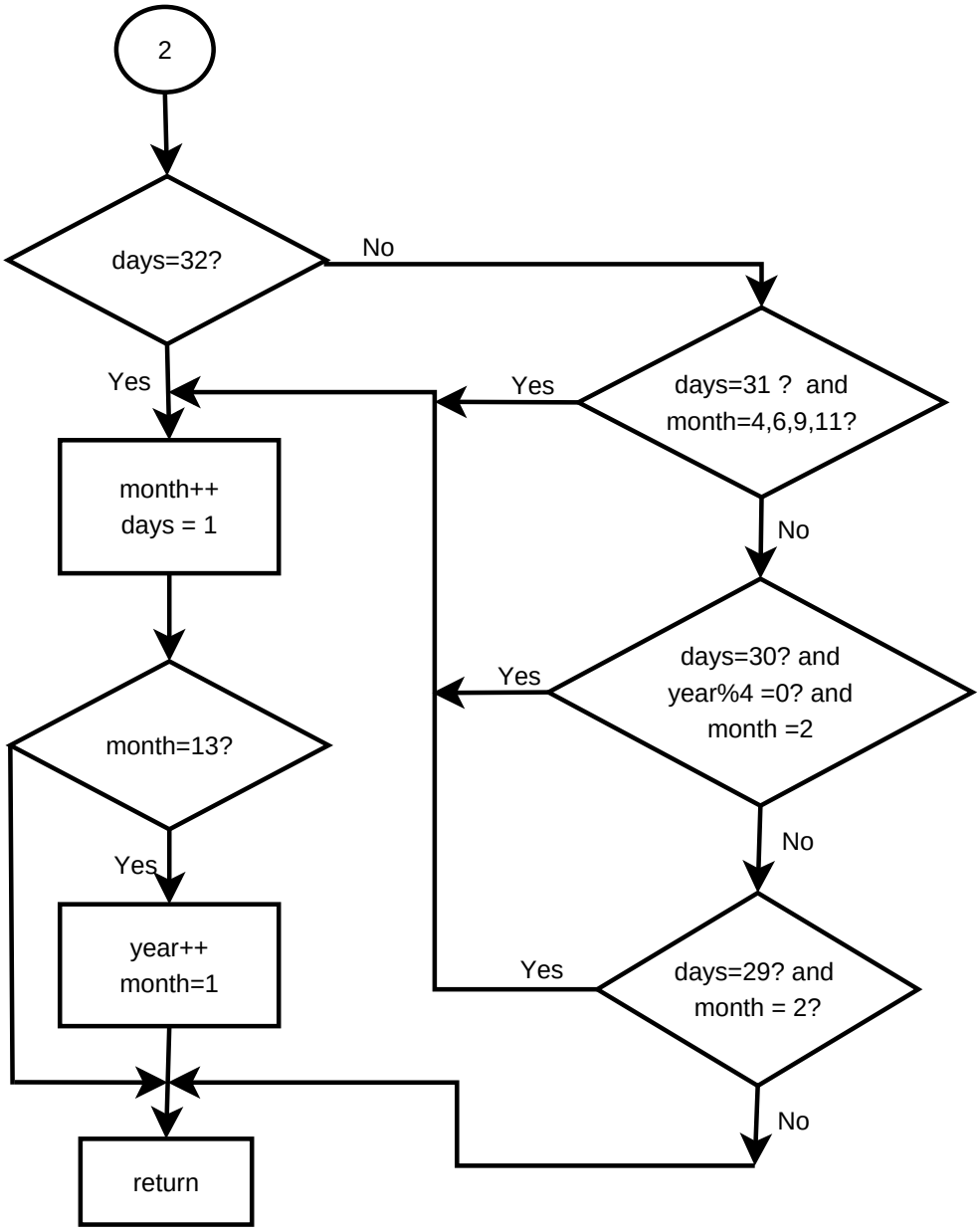


Figure 5.12: Continuation of the real time clock flowchart showing how the date is updated.

information. Even though a string like "start of file" is more human readable, numbers take up less space and are much easier to code for than strings. In any case, the data will be parsed by a GUI before the user sees it. It is much easier, cleaner and faster to code for number recognition, than string pattern recognition. The numbers chosen as markers were outside of the 0 to 1250 nA range of the test data so as not to be mistaken for data. The start of file marker is 11111. The header information is separated from the data with the numerical marker 21111.

Table 5.5: MiniEC test data file format

Address byte	Meaning
0	Start of file marker
2	file ID
4	Potential in mV
6	Interval in seconds
8	Duration in minutes
10	Test start time (year)
12	Test start time (month)
14	Test start time (day)
16	Test start date (hour)
18	Test start date (minute)
20	Test start date (second)
22	Start of test data marker
24	[Test data]
..	:
last	[Test data]

The number of bytes used by the test data depends on the recording interval and duration. For a basic test from a single sensor for a duration of 60s at 1 second intervals, the test data uses 120 bytes of memory. The total file size including the data descriptors in the header is 144 bytes. With data memory of about 40K, the miniEC can store over 100 of these files.

The development of a full-blown flash file data system was beyond the scope of this project, so we looked at incorporating an already developed file system. Many of the existing flash file systems available required several kilobytes of RAM

making them unsuitable for the MSP430 which has only 2 KB of RAM. Two flash file systems, ELF and Matchbox developed for the TinyOS sensor network nodes were thus investigated. However, it was concluded that given the modest data storage requirements, using any of such systems at this time was unnecessary. Instead, we developed a very simple scheme to store the data. The data storage scheme consists of the data files themselves stored in flash memory and three variables in RAM that keep track of the number of files, amount of memory used, and a pointer to the next available address in flash memory for file storage.

Each time the user requests a new test run, the system checks to see if there is enough memory to store the file based on the recording interval and duration selected. If there is enough memory, the data collected is written sequentially starting at the next available address. After each successful test run, the number of files, memory used and next available address pointer variables are updated. Each test data reading is written as soon as it is collected so in the event of power loss during a test run, data already collected are retained. The RAM variables are lost in a power outrage but are easily recalculated by going through the files already in the data partition, noting where the markers are.

Once the memory of the miniEC is full, measurements can still be made and viewed or transmitted to a PC in real-time but no data is logged. The miniEC will not automatically delete any files. Instead the user is notified of the situation and prompted to delete files when convenient. No attempt is made to even out the wearing of flash memory by this simple filing system. The MSP430 is guaranteed for 10,000 write-erase cycles and up to 100,000 cycle if used within the limits of commercial application environments. Thus, even if the user erases the whole data partition 10 times a day, the memory will still be fine for three years worth of operation.

CHAPTER 6

GRAPHICAL USER INTERFACE APPLICATION: DESIGN AND IMPLEMENTATION

6.1 Analysis

The miniEC GUI application is the software entity that facilitates the interaction between a user and the miniEC hardware using the familiar environment of a desktop PC or PDA. The operations that need to be implemented by the GUI for this task are:

- transmit and receive data from the device.
- configure device parameters
- provide basic analysis of recorded data
- provide storage of data and export functions for further analysis in third part programs

For data configuration of the miniEC device, data transmission and receipt, the GUI needs a device driver to communicate via either either an RS232 serial port, USB port or both. Basic analysis of recorded data requires that the GUI have the ability to plot simple graphs. Finally it must store the data in a file format that is easily interpreted by more powerful third party analysis tools or supply a method for exporting data to those programs.

The relatively independent nature of the tasks required provide natural divisions for modular code development. Figure 6.1 shows the components derived. The three main parts are the driver module that includes all the functions required to communicate with the miniEC hardware; a display module that graphically displays data and provides the visual widgets for a user to interact with the miniEC

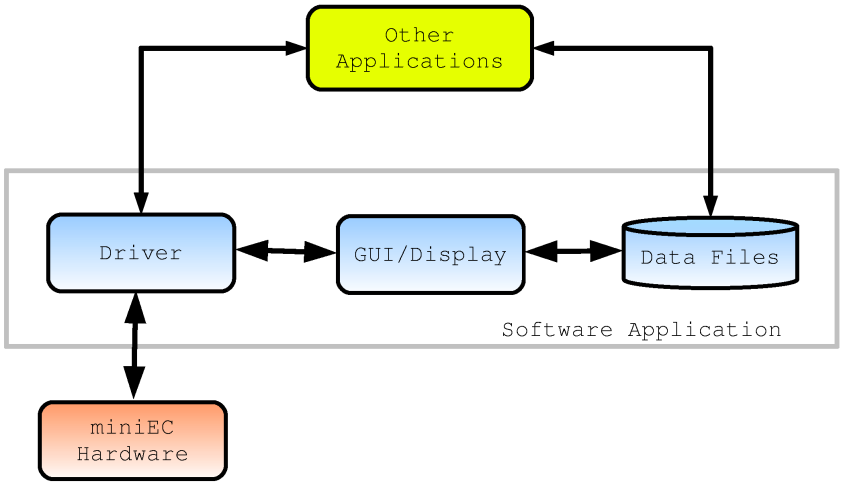


Figure 6.1: Components of the miniEC software application in relation to the device hardware. A driver module mediates the interaction between the a graphical user interface (display), a data storage module and the hardware. With the modular architecture, plug-ins developed by third parties can easily extend the functionality of the basic software with, for example, a statistical data analysis component or a better display.

hardware; and a data management module to organize and store data in a standardized format. The arrows between the modules show the direction of data flow. Data flows linearly from the miniEC through the driver module to the display and then the data modules. The reverse path is also linear through the data, display and driver modules to the miniEC. This arrangement ensures maximum flexibility. For example, other GUI front-ends can be substituted for the current one using an interface that is compatible with the output of the driver module. Another example that is not as obvious is that the application file format can be changed at any time without affecting the rest of the system. The data module can even be enhanced independently to format data for export to any number of third party applications.

6.2 Choice of design tools

Before implementation of the GUI, a decision had to be made about the programming language to use, which GUI toolkit to use and how to provide plotting functionality.

ANSI C (and its object oriented variant C++) was first considered, as it was the language used to develop the the miniEC firmware. It is excellent for getting maximum performance out of hardware because it executes fast and allows exquisite control of memory usage. For the miniEC GUI however such considerations were not paramount. Reliability and data integrity was considered more important. Thus, with the higher risk of memory leakage problems, potentially unstable or insecure code caused by buffer overuns etc., we concluded that a language with its own built-in memory management was essential. After considering three such languages – perl, python and java, perl along with the perlTK GUI toolkit was initially selected due to familiarity with the language. While this decision allowed rapid code development, the author was also familiar enough with perl to recognize the limitations of the prototype. The main problem with the perl prototype was that some hacks that had been added to work around a number of problems would make the code hard to maintain for another party.

Python was ultimately selected for a variety of reasons but primarily for the fact that Python code stresses readability and simplicity. The obviousness in syntax is important for other developers to successfully maintain, patch or modify the software application. Furthermore, unlike java, python is an open source product unencumbered by royalties or licensing fees. The literature also indicates that python is probably the only language that is truly cross-platform, as it runs on all the well known operating systems and on less well-known platforms, including PDAs and set-top boxes.[161]

Matplotlib was selected for the graphing tasks as it was by far the best 2D plotting library for python reviewed. It is also free, cross platform and produces publication quality figures in a variety of formats. Finally, WxPython was selected over the de facto python GUI toolkit, Tkinter. Both are cross-platform open source tools. Unlike Tkinter however, wxPython does not draw the GUI components. It simply provides an interface to the native GUI of whichever operating system it is running on, so applications have a native look and feel. This consideration is important for the acceptance and ultimate commercial success of the miniEC.

6.3 Implementation

The miniEC software application was developed on GNU/Linux, and is known to run on Linux (Figures 6.3 to 6.7) and Windows (Figure 3.11) While it has not been tested on an Apple computer, the use of cross platform tools such as python and wxpython for the code and graphical interface suggests that it should also run with little to no modification on Apple OS X.

Figure 6.2 shows a simplified view of the implementation of the software application. In the tradition of code reuse emphasized by python, the application uses several standard and non-standard python libraries to simplify development. The gray boxes in figure 6.2 reference code developed for the project while the white boxes refer to existing code libraries. Both miniEC.py and plotting.py use modules from the standard Wx library for python to render the user interface. In addition, the plotting.py module uses the WxMpl library to embed plots in the wxpython user interface.

The miniEC.py script depends on methods from the the driver.py module to autodetect and communicate with a miniEC device. The driver.py module in turn depends on the pyserial library for methods that simplify communication between

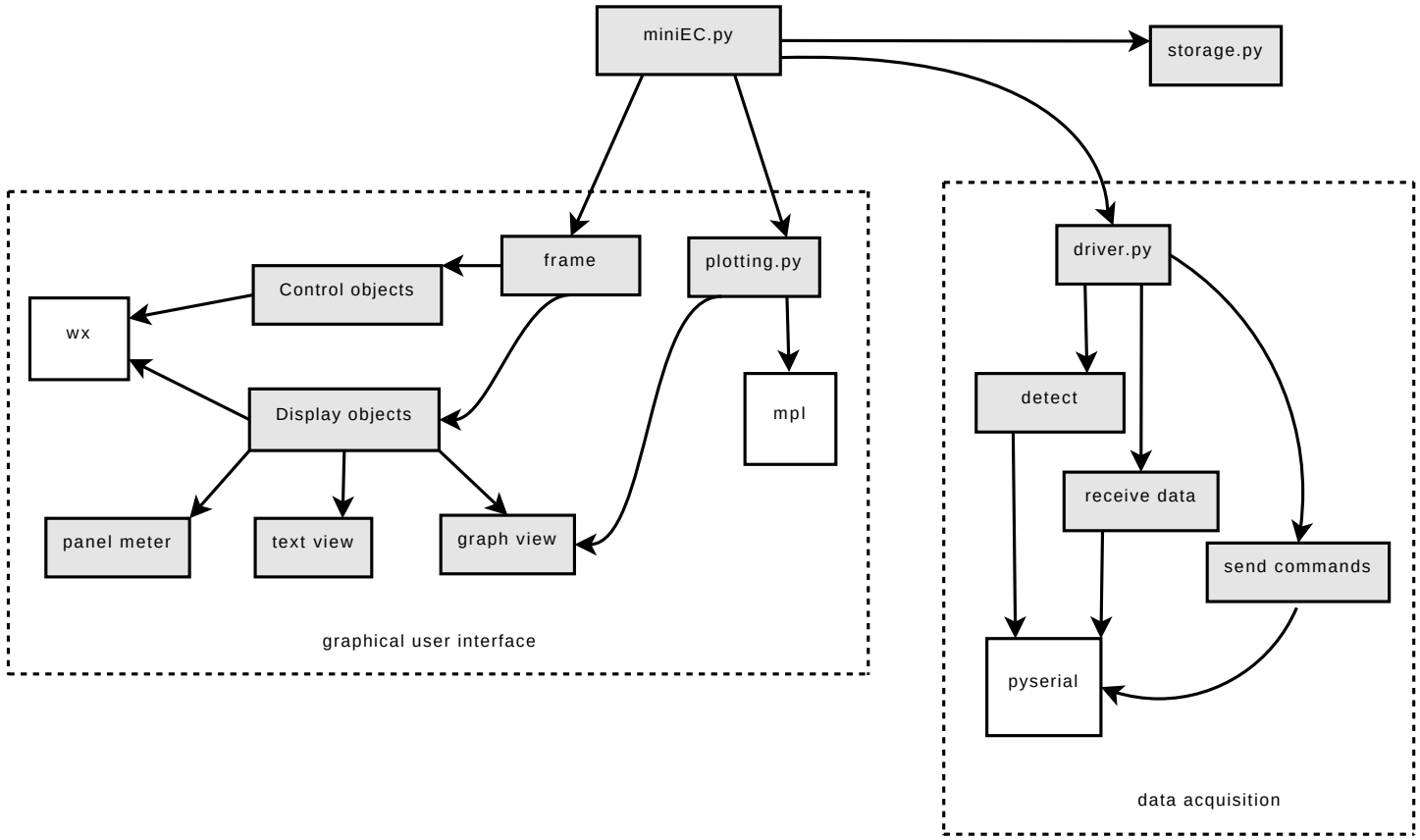


Figure 6.2: Simplied view of code dependencies between modules for the miniEC PC application. The gray boxes reference code developed for the project while the white boxes refer to existing code libraries. The graphical user interface is derived from control and display classes within miniEC.py and plotting.py. Plotting.py uses methods from the matplotlib (mpl) library and both use modules from the standard wx library for python. The driver.py module encapsulates methods to communicate with a miniEC device using the pyserial library. The storage module renders collected data into a structured XML format for export to third party applications.

the serial communication hardware on a computer and the miniEC. The pyserial library provides cross-platform support for serial connections with legacy RS232 serial ports, USB, bluetooth dongles, infra-red ports etc. Incidentally the pyserial library is also used by the mspgcc tool chain to program MSP430 microcontrollers.

MiniEC.py also depends on the storage.py module to transform the data into an XML format for export to third party applications. Other python library dependencies include the use of the scipy and numpy scientific and numerical toolkits to provide fast processing of data. A full listing of the code is provided in Appendix A.

6.3.1 Driver

The miniEC hardware may present two kinds of serial interfaces, RS232 or USB, for networking. The driver uses methods in the the pySerial module library to detect RS232 connections directly and a USB connection as a virtual RS232 serial port. Consequently, both interfaces are accessed in the same way.

The driver contains 3 classes: miniEC_options, miniEC_status and miniEC_comm. The miniEC_options and miniEC_status classes are holders for amperometry settings and information about the status of a miniEC device. The miniEC_comm class encapsulates the methods used to communicate with a miniEC device. These methods and a description of what each does are given in Table 6.3.1.

In addition to the methods in Table 6.3.1, the script can be run in an interactive python environment independent of the GUI. Three other methods comm_expt_start, comm_expt_run and comm_expt_stop are provided to facilitate this mode. In this mode data is displayed in the stderr/stdout console.

Table 6.1: Methods provided by the miniEC_comm class to interact with a miniEC device

Method	Short description
comm_find_device	Scan ports for a miniEC device
comm_open	Open connection to a miniEC device
comm_close	Close connection to a miniEC device
comm_flush	Clear send and receive buffers
comm_send_command	Send a command to a miniEC device
comm_read_datapoint	Get result for one measurement interval
comm_get_data	Download all data stored in on miniEC device
comm_get_data_headers	Download a summary of stored data
comm_clear_data	Delete all the data logs stored on the device
comm_get_status	Get the status of a miniEC device
comm_update_options	Upload settings to a miniEC device
comm_get_time	Get date and time reading from a miniEC device
comm_set_time	Set the date and time on a miniEC device

6.3.2 GUI front-end

The graphical user interface (GUI) provides point and click access to the driver methods discussed in the previous section for interaction with a miniEC device. There are three main windows in the GUI for data visualization: a meter panel format showing only the last value measured, a text view showing a list of all values and a graphical plot of all values. The meter panel is embeded into the main frame of the application window which also contains various buttons and menu options for program control.

Figure 6.3 shows the meter panel view which is the first window shown after

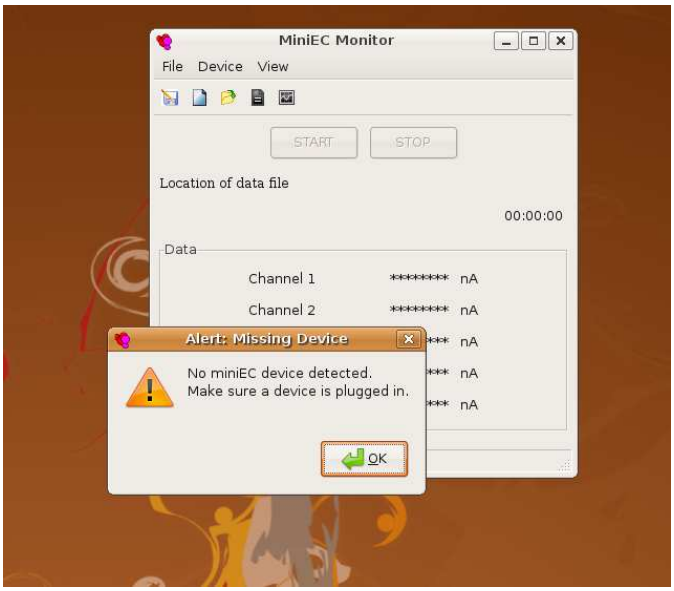


Figure 6.3: The MiniEC Monitor meter view showing the message dialog displayed when a device is not connected.

launching the miniEC monitor executable. During the launch, a background process using the driver module will attempt to connect to a miniEC device. If one is found, the settings and status menu dialogs will be populated with values from the device. Since the GUI can also be used to visualize already recorded data, it will not abort if a device is not found, but simply report the absence of a connected miniEC device and wait for further user input.

To intiate an experiment, the user must select "New" from the file menu and provide a filename to save data to. Completing this step enables the "Start" button. Clicking the start button starts the experiment. The current recorded for each channel is displayed in the panel meter. A user can also select the text or graph options from the "View" menu or click on the corresponding buttons in the main window toolbar to view the data. Figure 6.4 shows a meter panel in a typical measurement session with associated text and graph views. The main window toolbar has 5 tools for the "save as", "new', "open', "view text" and "view graph" actions in that order.

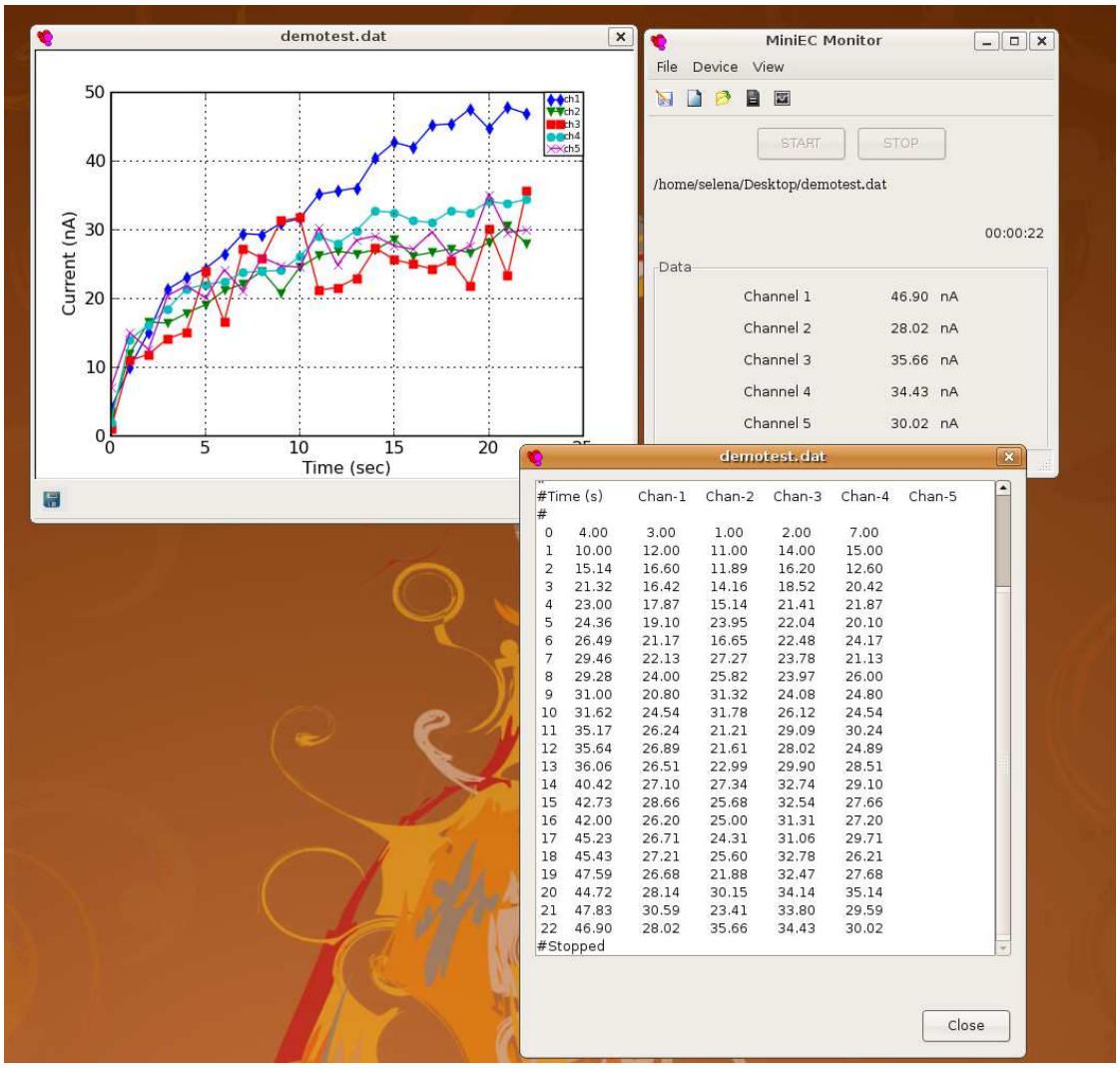


Figure 6.4: A screenshot of the three data presentation views of the miniEC monitor. Data can be viewed as text, in a graph or as data points in the meter view.

The text view currently has no tools for the manipulation of its contents. The graph can however be zoomed, resized, copied and saved. In addition, coordinates of any point on the graph indicated by the mouse curser is displayed on the bottom left hand corner of the figure. To zoom in, simply select a rectangular area of the graph by clicking and then dragging the mouse across the desired area. Zoom out by right clicking the mouse. A save button located in the bottom left hand corner of the graph view toolbar enables the user to save the figure to a variety of formats

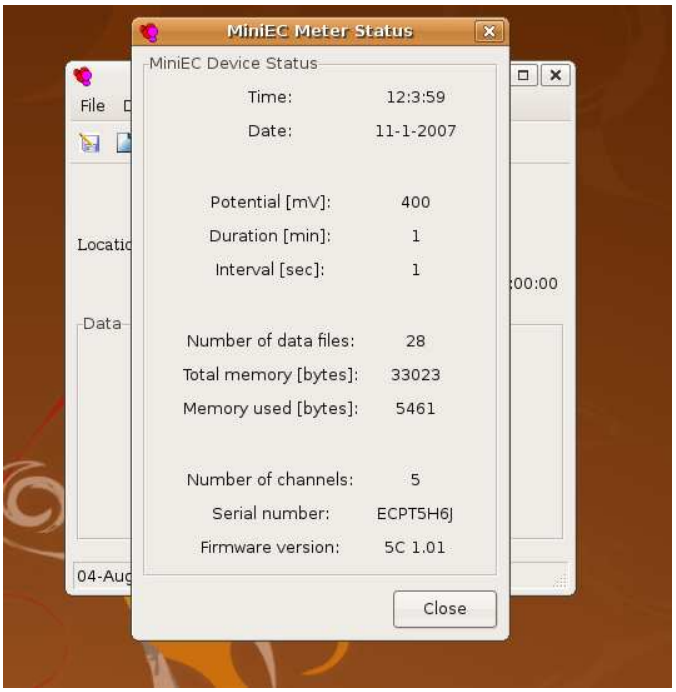


Figure 6.5: A screenshot of the status of a connected miniEC meter

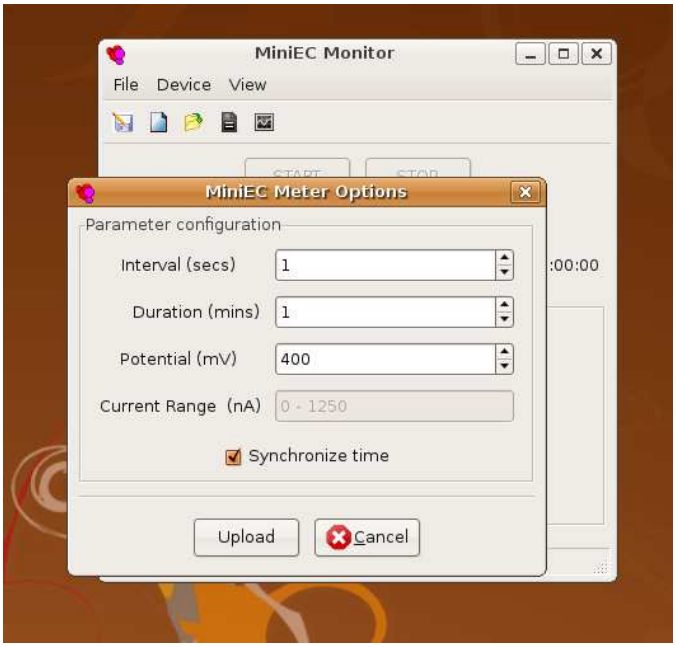
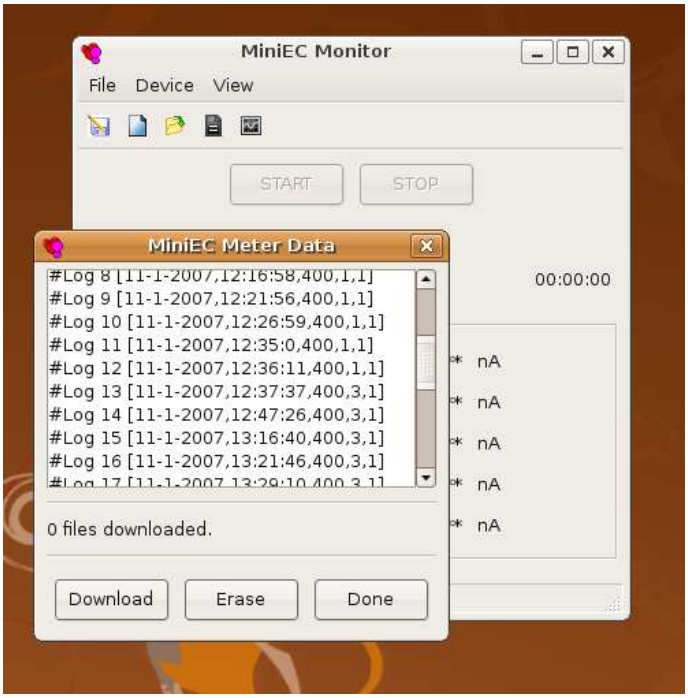
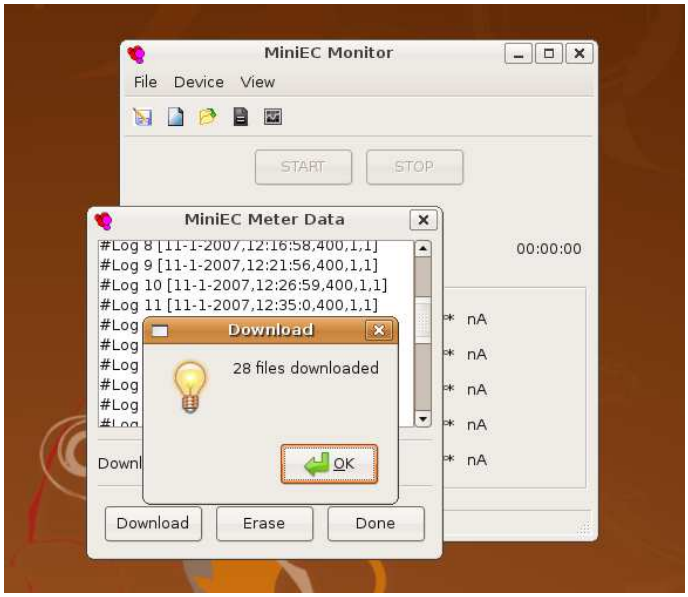


Figure 6.6: A screenshot of the options dialog to set the time and amperometry options on a device.



(a)



(b)

Figure 6.7: The data dialog for viewing, downloading and erasing data from the memory of a miniEC device. Figure (a) shows a list of logs available on the device for download. Figure (b) shows confirmation of files downloaded after clicking the "Download" button.

including PDF, PNG and EPS.

By default data is saved during an experiment in a simple CSV format with a space delimiter. After an experiment, the user can use the "save as" option to rename the data file or export the data to an XML format. Previously recorded data can also be opened and saved to XML in the same fashion.

The user may view the status of a device or update settings by selecting the "Status" or "Options" items from the "Device" menu respectively (Figures 6.5 and 6.6). The "Data" option in the device menu provides a data dialog (Figure 6.7) where the user may view a summary of the logs stored on the miniEC device, download the data logs or erase them from the miniEC's memory.

6.3.3 Data storage

One of the most time-consuming tasks for software developers is writing code to exchange data between applications that may have poorly documented, and proprietary, or both kinds of file formats. More often than not, this is the result of a delibrate decision by developers as part of a vendor lock-in strategy. In contrast, the miniEC data storage uses simple text files in CSV and XML formats identified by .dat and .XML extensions respectively.

The CSV format simply prints out the metadata of an experiment on lines preceeded by the hash symbol "#" followed by data in columns. A tab space is used as the delimiter between columns. The first column lists time in seconds. Subsequent columns hold data from a single channel, 1 channel for the basic miniEC device and 5 channels for the multi-sensor device. Thus each row represents data read off of the miniEC at the same time. Any comments generated during the running of an experiment and stored in the file are also preceeded by hash symbols. Third party software programs for data analysis can thus be instructed to ignore all lines

that begin with the hash symbol and process only those without.

While the CSV format has been time tested and widely used for data transport and transformation between systems, it is now considered somewhat of a legacy format. It is being replaced by XML or the eXtensible Markup Language. As with the CSV format described, the data creation parameters are stored with the data in flat files in the XML file, so both formats provide a software and hardware independent way of storing data. The content representation in an XML document is however made more explicit by enclosing data within descriptive tags that may be nested to show relationships between the types of content. The legal structure of an XML document is defined by a document type definition (DTD) or an XML schema. Both define the document structure of an XML file with a list of legal elements against which a third party application can check a document to ensure that it is valid. The XML schema allows a more stringent definition of what is permitted than a DTD. The miniEC DTD and schema are listed in the Appendix. An example miniEC XML file is shown in listing 6.1.

The miniEC XML format has a top level element tagged <miniec>. Under this, are 3 main elements <head>, <extra> and <ecdata>. The <head> element is populated by required metadata information from the creation of the data file. The <extra> element contains metadata information that is currently deemed optional. The <ecdata> element contains the time series and actual current values measured at every interval in the time series.

The advantage of XML over CSV is in the ability to easily parse and index archived data from several files by specific criteria to discover patterns using standard XML software tools. The same XML structure can be mapped to a database providing a user with even more tools for data discovery. This will become par-

ticularly important in a multiple user, multiple location application where many references will be made to who, where and how the data was collected. Referential constraint mechanisms built into an XML or database structure will help researchers develop and manage such applications. Another advantage of XML as the export format is that it provides a clean well-documented structure for automated transformation into other file formats.

Listing 6.1: A small miniEC XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<miniec>
    <head>
        <experiment>DCPA</experiment>
        <date_stamp>11-1-2007</date_stamp>
        <time_stamp>12:0:50</time_stamp>
        <interval units='sec'> 1 sec</interval>
        <duration units='mins'> 1 min</duration>
        <potential units='mV'> 400 mV</potential>
        <range units='nA'> 1250 nA</range>
        <channels>5</channels>
    </head>

    <extra>
        <note></note>
        <user></user>
        <test_label></test_label>
        <software></software>
        <firmware></firmware>
        <serial_id></serial_id>
    </extra>

    <ecdata>
        <time units = 'sec'>0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0</time>
        <channel units = 'nA' number = '1'>1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</channel>
        <channel units = 'nA' number = '2'>1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</channel>
        <channel units = 'nA' number = '3'>1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</channel>
        <channel units = 'nA' number = '4'>0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</channel>
        <channel units = 'nA' number = '5'>0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0</channel>
    </ecdata>

</miniec>
```

The storage.py module handles the formatting of the metadata for the .dat files and the transfomation of the default .dat format to XML. It contains two template functions, one for the .dat file header and the other for an XML document

that conforms to the structure described by the miniEC DTD. Each template has placeholders enclosed within dollar signs, like $time_stamp_here$, which are replaced by the actual values when the document is rendered.

CHAPTER 7

DESIGN FOR A MULTI-SENSOR MINIATURIZED ELECTROCHEMICAL DETECTION SYSTEM

One of the often cited advantages of microfluidics is the potential for rapid high throughput analysis at lower cost compared to traditional methods. This advantage has been exploited in many cutting edge laboratory instruments to speed up processes such as gene sequencing and medical diagnosis. It is however underexploited in the few portable analysis systems (mainly glucose monitors) that exist on the market today.[162]

This chapter provides an example of how to extend the basic miniEC previously developed in Chapters 3 to 5 into a multi-sensor miniEC device. As with the basic miniEC, this multi-sensor miniEC is in two parts, consisting of disposable sensor cartridge and a reusable electronic control unit. The new prototype has the capability of running up to 5 electrochemical tests in parallel but retains the small size, low-power, low cost, and high performance of the single test system.

The biosensors used with the 5 channel miniEC are identical to the basic miniEC model sensors. A single sensor consists of an IDUA transducer that can be mated with a microfluidic channel or used alone in a non-stirred assay. Up to five of these individual sensors or a single cartridge with up to five separate electrode arrays in individual microfluidic channels may also be used with the multi-sensor miniEC electronic control unit.

7.1 Hardware Design

The functional block diagram of the multi-sensor MSP430-based miniEC device is shown in Figure 7.1.

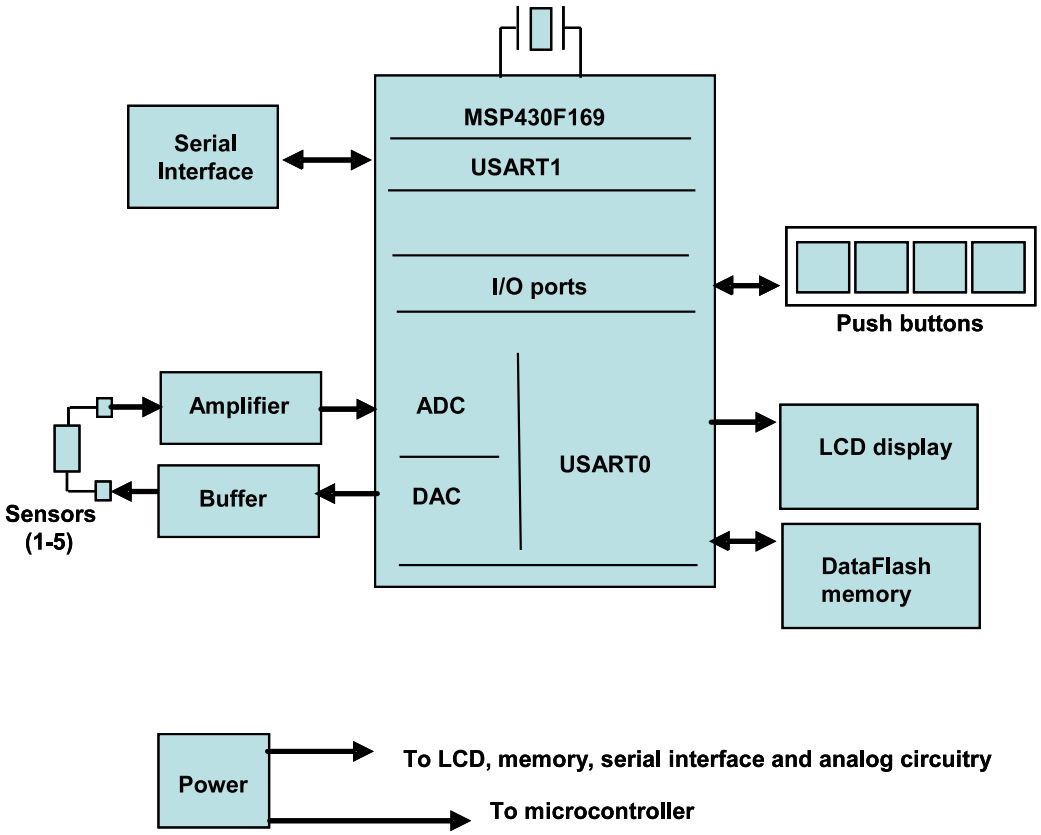


Figure 7.1: Block diagram of the MSP430 based multi-sensor miniEC system. A DAC output from the microcontroller generates the potential to bias the sensor. Up to 5 sensors may be connected. Each sensors output is converted to a voltage and amplified to a range that can be read by the microcontrollers analog-to-digital converter (ADC). The results are displayed on the LCD and a copy stored in the dataflash memory for later retrieval via the serial interface. The major difference between this and the basic miniEC (Figure 4.3) is the substitution of the MSP430F169 for the MSP430F439 microcontroller and the addition of dataflash memory to accomodate data from multiple sensors. Other differences are discussed in the rest of this section.

Similar to the basic miniEC the multi-sensor miniEC system consists of a set of five modules each with its own dedicated responsibility. It has a connectivity module for serial communications; a power supply module; an analog module that conditions the signals exchanged by the sensors and microcontroller; a digital module that contains elements of the user interface and memory for data storage; and finally the core microcontroller module that ties everything together.

7.1.1 Core MSP430 module

The MSP430 microcontrollers in general have a wide range of features and peripherals and they supports low-cost JTAG hardware for programming and debugging. Furthermore, their power consumption is an order of magnitude lower than comparable microcontrollers currently available for portable instrumentation design. It is also relatively easy to substitute one microcontroller for another especially when programming is done in a high level language like ANSI C.

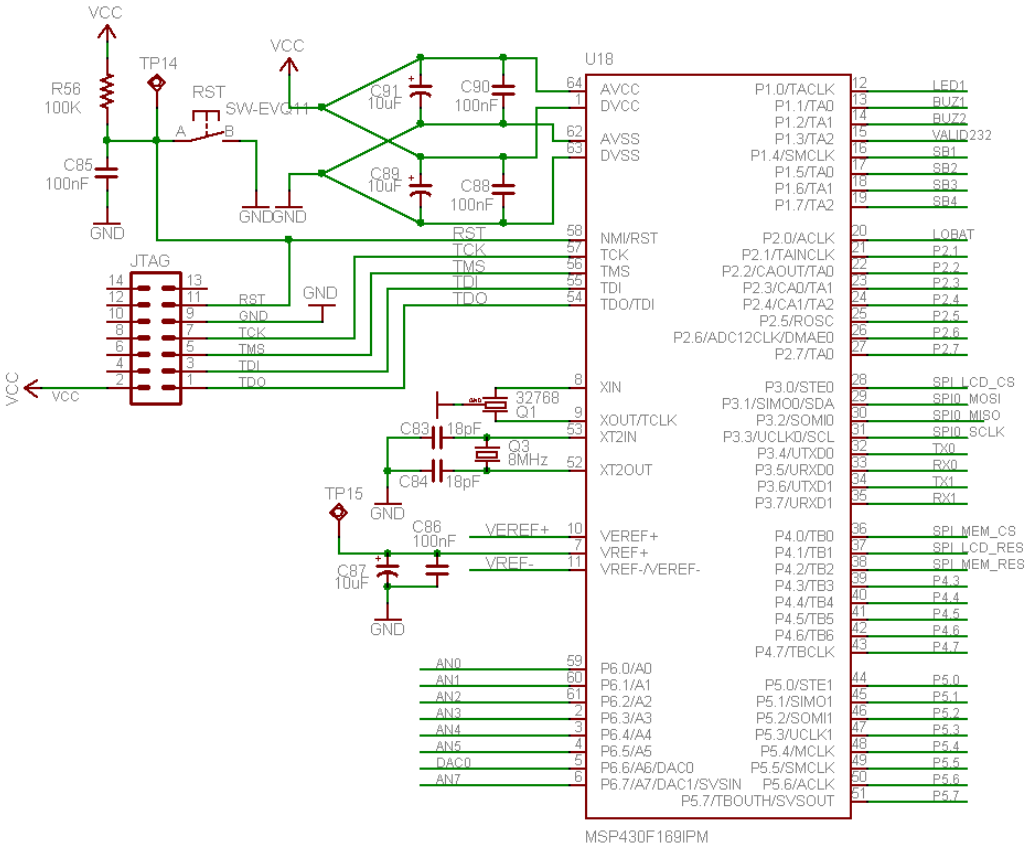


Figure 7.2: The core MSP169 module showing how the microcontroller is connected to other elements of the system.

The MSP430FG439 microcontroller in the basic miniEC was replaced with an MSP430F169 for the multi-sensor device for a number of reasons. The F169 has

2 USART peripherals for serial communication while the FG439 had one. The multi-sensor microcontroller needs to implement 2 different serial communication protocols, one synchronous and the other asynchronous. The synchronous serial peripheral interface (SPI) protocol is needed to communicate with the LCD and dataflash memory. The asynchronous protocol is needed to implement a minimal RS232 protocol used for USB and serial port communications. The MSP430F169 also does not have the 3 internal op-amps or the special function LCD control outputs of the FG439 and so, has more general purpose digital input/output pins available. All other parts are identical between the microcontrollers, consequently, most of the code from the single sensor miniEC was reusable for the multi-sensor device.

Apart from the power and JTAG interface, the MSP430F169 microcontroller has 6 ports (Figure 7.2). Each port contains 8 individually programmable I/O pins. Port 1 is configured in this design to control the output of the buzzer and status LED. It is also configured to receive input from the 4 pushbuttons and the MAX3221 serial communication chip. Each time a button is pressed or a serial port connection is made, there is a change in the potential of the inputs. This change triggers an interrupt in the microcontroller software so that it executes the appropriate response to the trigger event. Pin 1 of port 2 is also an input from the power module that changes state to alert the microcontroller when the battery is low. Pin 8 of port 2, is the USB power good signal that indicates when a USB connection is made to a PC. The rest of the port 2 pins are unused.

The MSP430F169 implements a serial peripheral interface protocol (SPI0 SCLK, MISO, MOSI) with its first USART module on port 3. The LCD and dataflash memory are the digital peripherals controlled by SPI. The two peripherals receive and send data on the same SPI bus. When the microcontroller needs to communi-

cates with the LCD, it indicates the selection on the SPI_LCD_CS wire. Likewise if the communication is meant for the dataflash, the microcontroller enables it via the wire labeled SPI_MEM_CS. The microcontroller can also reset the two SPI peripherals through the SPI_MEM_RES and SPI_LCD_RES wires on port 4.

The second USART, also on port 3, is used to implement serial RS232 communication to a PC's serial or USB port using just 2 lines (TX1, RX1 in Figures 7.3 and 7.4). Port 5 is unused. The analog and digital converters are located on port 6. The MSP430F169 has 2 digital-to-analog converters, one of which (DAC0) is used to produce the sensor bias potentials. Five of its analog-to-digital converters (AN1 to AN5) are used to measure and convert amplified current from 5 IDUAs into digital values that are subsequently stored in the dataflash memory.

To keep all parts and of a microcontroller system synchronized, the CPU performs all operations in a timed sequence determined by a clock. Here, a clock is a circuit that produces a periodic signal, usually a square wave with a known frequency. Since digital transitions occur either at the rising or falling edge of a clock signal, the higher the clock frequency, the more operations a microcontroller can perform in a given time. High frequency operations however consume more power and are also more expensive. MSP430 microcontrollers can generate their own clocks in a more power efficient way than external clocks. For the greatest accuracy in synchronization however, at least one external crystal oscillator is needed to generate the clock signal. Two oscillators; a low frequency 32 KHz and a high frequency 8 MHz; are provided to time the execution of instructions on the miniEC. In this application, The 32 KHz one is always on and used for the real time clock and serial communication. The 8 MHz is only used by the CPU intermittently for computationally intensive tasks.

The core MSP430 module also includes a reset button (RST) that may be

pressed to "restart" the miniEC should it malfunction during operation or whenever the user desires to restore the default parameters. The rest of the microcontroller's pins are for the power and JTAG connections.

7.1.2 Connectivity module (serial interface)

The connectivity module of the multi-sensor miniEC retains the serial port circuitry of the miniEC and adds USB circuitry. The user now has the option to use either an RS232 serial port or a USB port for serial communication with a PC.

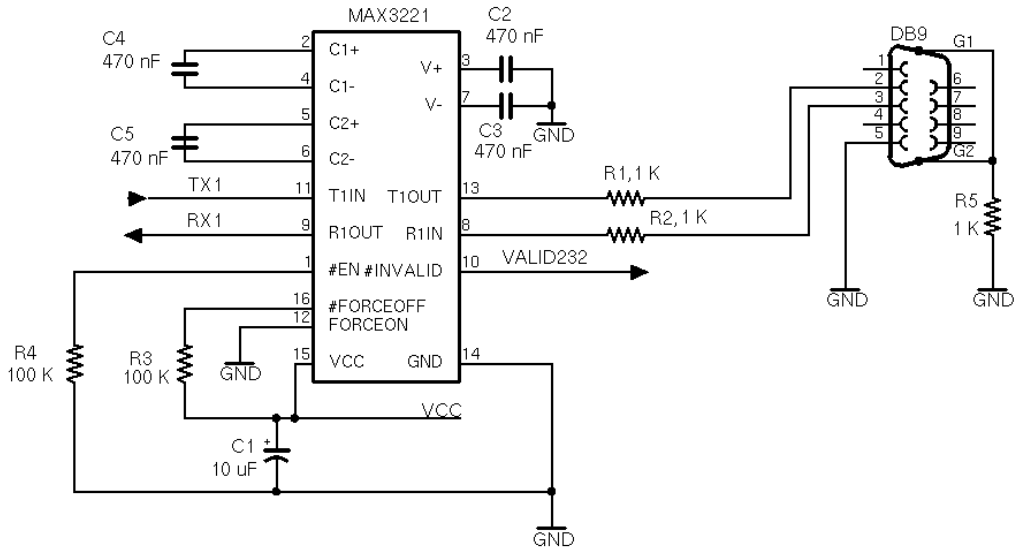


Figure 7.3: The MAX3221 interface for connection to a PCs serial port. It receives and transmit to the microcontroller via the RX1 and TX1 lines respectively. The VALID232 wire can be used to notify the microcontroller when the miniEC first connects to a serial port.

The MSP430F169 microcontroller used in this application is equipped with two built-in USART peripherals. Both have the capability to easily implement a serial communication protocol. Only two I/O pins, TX and RX, and a ground connection are required to implement a minimal RS232 serial interface. The microcontroller however cannot be connected directly to a PC's serial port because of the difference

in power requirements. The MAX3221 level shifter is retained from the basic miniEC design to translate the 3 V levels of the microcontroller's signals to 12 V levels required for RS232 transmission. It can operate from a low supply voltage of 3 V, has low power consumption, a low standby current mode and automatic power-down features making it ideal for battery-powered portable equipment.

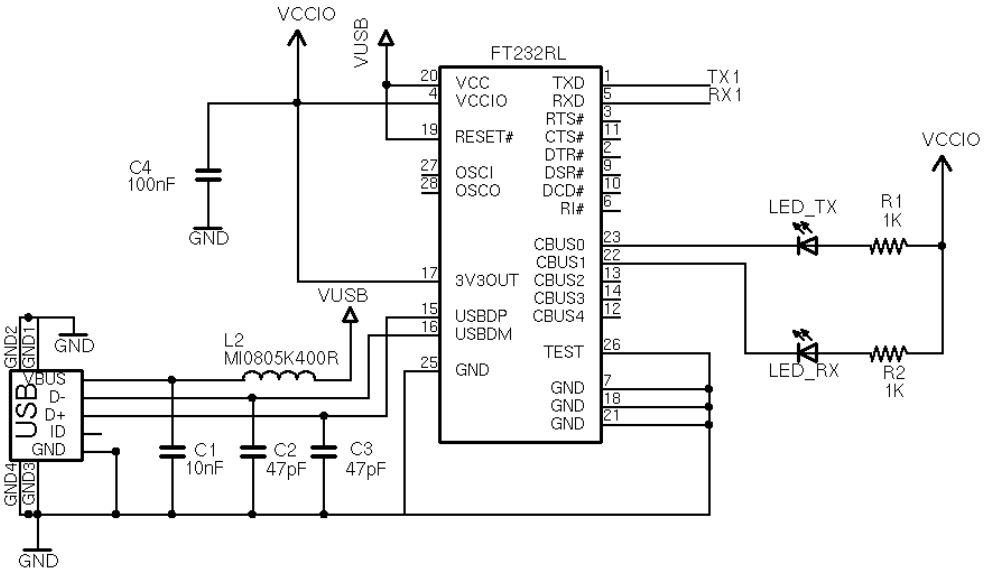


Figure 7.4: The minimal FT232RL interface for USB connectivity. It is powered by the PC and provides a power source, VUSB, to the rest of the circuitry when the miniEC is connected to a USB port.

Figure 7.3 shows the schematic of the MAX3221 chip as it is used in the miniEC. With FORCEOFF tied to VCC and FORCEON tied to ground, the auto-powerdown feature of the MAX3221 is enabled in this application. In this mode, the MAX3221 uses a supply current of just 1 μA until it senses a valid RS232 signal. It then resumes operation in its normal mode until the serial transaction is complete. The VALID232 output transitions from a logic low to a logic high voltage when the MAX3221 exits powerdown mode. This signal is used to notify the microcontroller when the miniEC is connected to a serial port.

Today, the RS232 serial protocol has been replaced by USB on computers as

the de facto standard for general purpose serial communications. In fact, most modern computers do not ship with serial ports anymore. Keeping the serial port on the miniEC however allows it to be used with legacy PC systems and other serial devices such as mobile phones and PDAs that are more prevalent in low resource environments. In future configurations of the miniEC, the RS232 electronics can be replaced with a wireless communication module.

With its plug and play attributes and superior transfer speeds, USB's ease of use from the end user's perspective is undeniable, however, adding USB support to a device is actually quite difficult since the interface is very complex. Figure 7.4 shows how the multi-sensor miniEC design avoids much of the complexity in USB development by using one of Future Technology Devices International's (FTDI) conversion products, the FT232RL. The FT232RL chip provides a fully integrated circuit that minimizes board space and performs RS232-to-USB bridging functions to and from the microcontroller's USART1 to the user's PC. FTDI supplies all the required software drivers royalty–free. As a result, these drivers have now been incorporated into all recent versions of desktop operating systems. The system can therefore be used with RS232 and USB equipped computers with few changes in the end user software.

7.1.3 Power supply module

The enhanced miniEC, like the basic model is required to run off a 3V power supply. With more components such as a graphic LCD and dataflash memory drawing power, the battery pack was doubled to 2 AAA batteries and the TPS60310 voltage regulator swapped out for a TPS60204. The new power module design also takes advantage of the fact that power can be obtained from the USB connection to conserve battery power when the miniEC system is connected to a desktop PC.

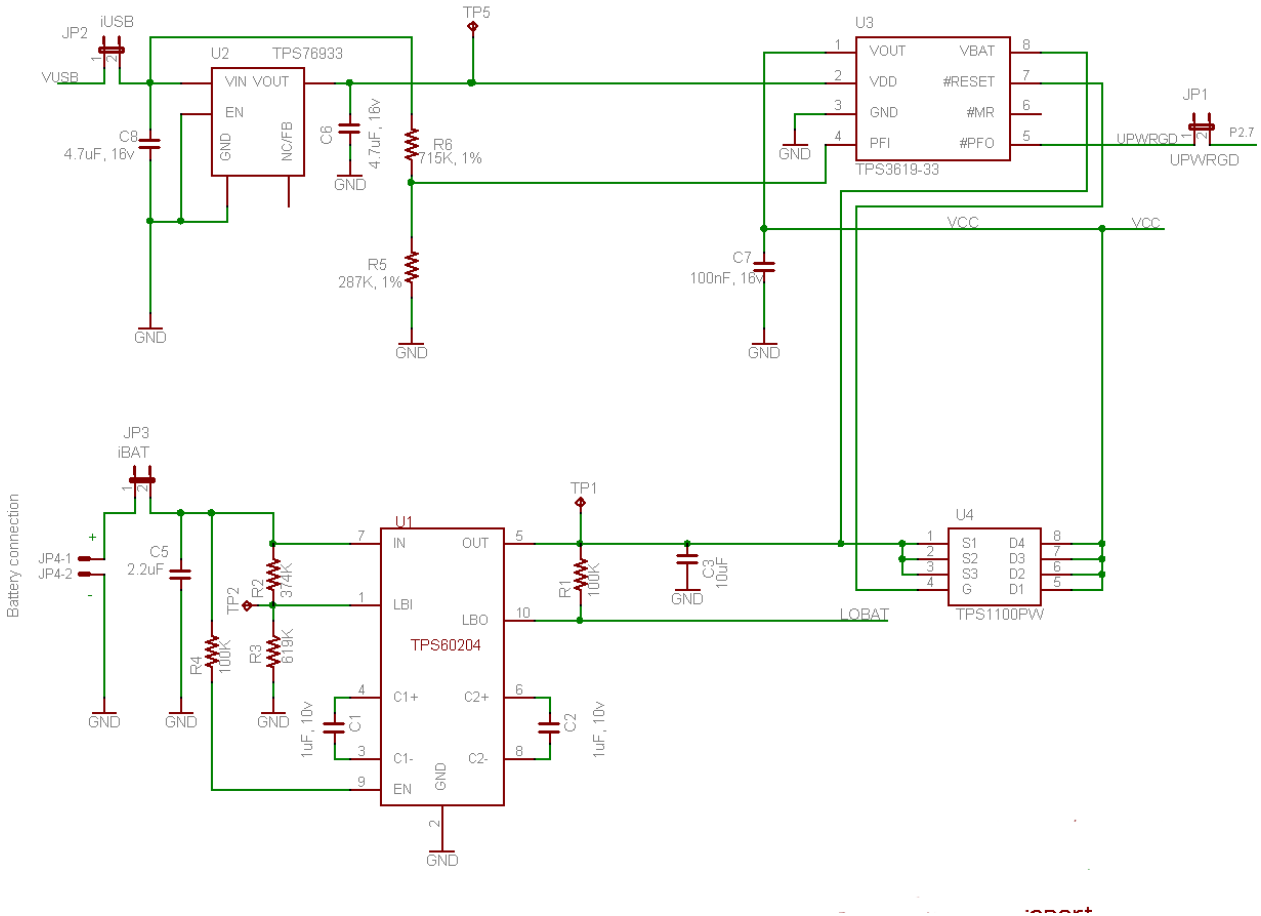


Figure 7.5: The power module.Two voltage regulators,TPS60204 and TPS76933, provide power sourced by batteries and USB port respectively. Battery power is normally conducted to the rest of the circuit through the TPS1100 switch. When the miniEC is connected to a USB port, The TPS3619–33 battery supervisor disables the TPS1100 switch which disconnects the battery so the system can be USB powered.

Power from the USB bus and battery are converted to the 3V positive supply voltage (Vcc) level required by the system with the TPS76933 and the TPS60204 voltage regulators respectively (Figure 7.5). The power switching between the 2 sources happens automatically without user intervention. In the absence of a USB connection, battery power is conducted through the TPS1100 switch to the rest of the circuit. When the miniEC is connected to a PC via USB, the connection is sensed by the TPS3619–33 battery supervisor on its PFI input. The battery supervisor disables the TPS1100 switch which disconnects the battery and the system is instead powered by the USB bus through the TPS76933.

In stand-alone miniEC operation, the TPS60204 can supply up to 100mA at 3.3V for system operation from a battery potential of 1.8 to 3.6V. In addition, the TPS60204 has a low battery output LBO, that can be set to warn the MCU to protect data and perform its programmed idle or shutdown routine. The voltage at which the low battery warning is issued is set by the voltage divider arrangement formed by R1 and R2 on pin LBI of the TPS60204 voltage regulator as shown in Figure 7.5. Using an R1 of 374K and an R2 of 619K, the TPS60204 will send a low battery warning to the microcontroller when the input battery voltage falls below 1.9V.

7.1.4 Analog peripherals

The analogue module (Figure 5) of the miniEC contains op-amp circuits that condition the signals between the IDUA sensors and the microcontroller. The circuit was built with the same low-power, low noise MAX407 (Maxim Integrated Products, Sunnyvale, CA) op-amps with the ideal characteristics of very high input impedance and a typical low input bias current of 0.01 nA were selected for the single-sensor miniEC. Instead of individual op-amp chips however, two MAX418

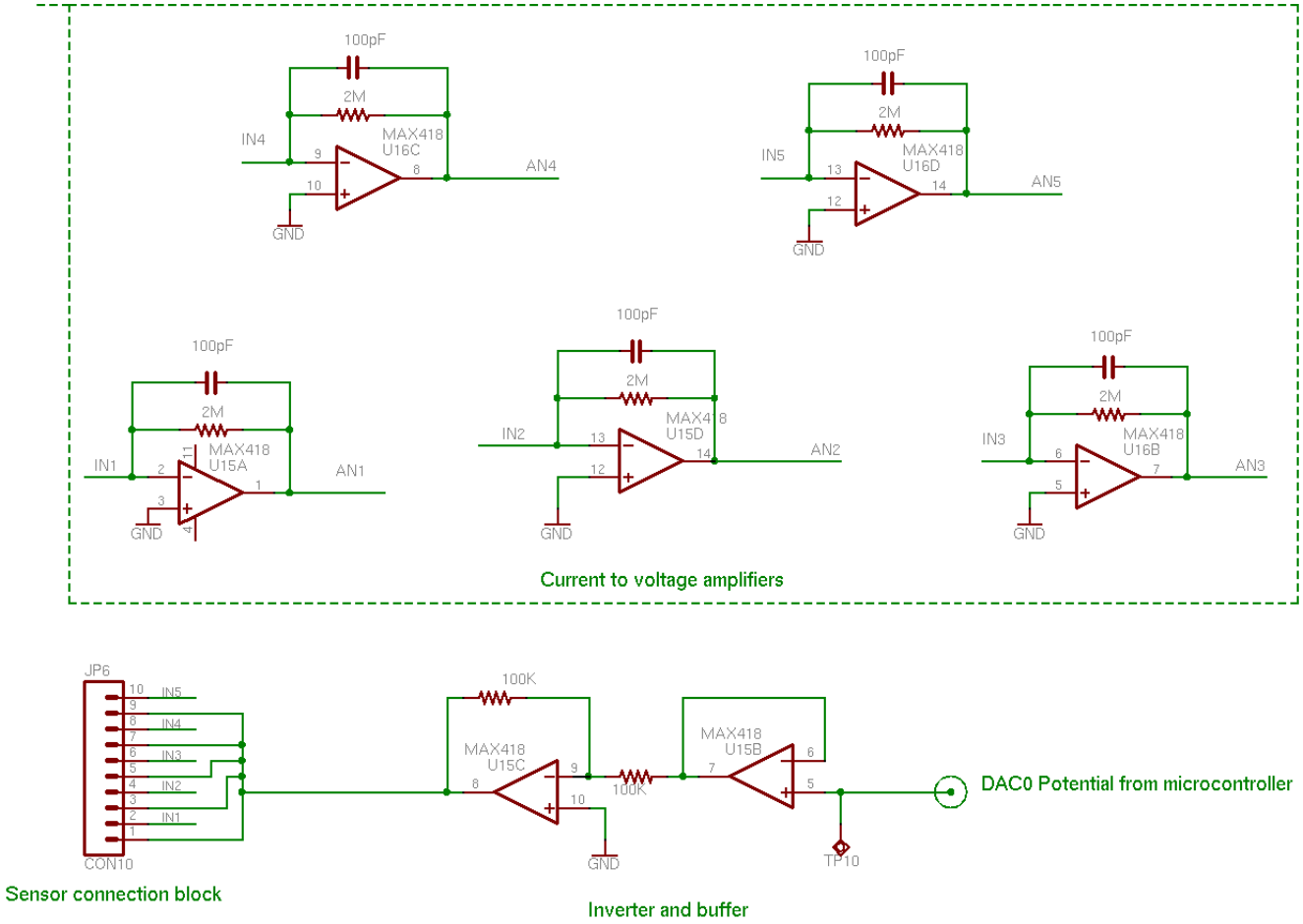


Figure 7.6: The analog module. The potential required to bias the IDUAs is produced by the microcontroller's DAC peripheral. The signal chains from the DAC to the IDUAs and from the IDUAs (IN1 – IN5) to the microcontroller's ADC (AN1 – AN5) form 5 potentiostats.

chips each with four MAX407 op-amps packaged into one chip were used. Five of the op-amps were used to amplify each sensors output in the same manner as the single sensor miniEC and two were used to condition the sensors bias potential.

To power the IDUA sensors electrode arrays the MSP430 generates a positive potential from an internal digital-to-analogue converter. This potential is inverted and buffered by two op-amps before applying it to each sensors reference electrode array. An inverting amplifier with a gain (R) of 2 MΩ on each sensors working electrode array converts current produced by the IDUA into a positive voltage within the measurement range of the microcontroller's analogue-to-digital converter which is 0 to 2.5 V. Thus, 1 nA = 2 mV and the current range measured by the ADC is 0 to 1250 nA.

Unlike the rest of the circuit, the op-amps as used in this module require a negative power supply in addition to the positive supply to perform the functions described above. Thus the third major component of the analogue module is a MAX1044 negative voltage converter which produces a local negative power supply (not shown in Figure 7.6).

7.1.5 Digital peripherals

The digital peripherals include an LED, buzzer, four push buttons and a liquid crystal display (LCD). These components form a simple menu driven user interface. System status, instructions, data and menus are displayed on the LCD. The user navigates the menus with the push buttons. The LED and buzzer are used to signal simple events and alerts to the user. For example, the LED will flash when the miniEC is first powered up and the buzzer will beep briefly when an experiment is complete. The buzzer is connected to timer A of the microcontroller on pins P1.1 and P1.2. The buzzer can only produce a single frequency. One of the

microcontrollers timers, however, is programmed to apply a pulsed signal that switches it on and off at different rates to simulate other audible frequencies for various system alerts. The four push buttons are connected to the microcontroller on pins P1.4 to P1.7 on port 1. An interrupt is generated every time one of them is pressed. Their functions and effect on the miniEC are defined by software (see Chapter 5).

The SBLCDA2 can only display one data point at a time as such it is not suitable for a multi-sensor miniEC system. Furthermore, the MSP430F169 does not have the segment control pins for it. A suitable replacement is the PCD8544 controlled graphic LCD. It is a low power LCD that measures 38 by 35 mm, with an active display area of 30 by 22 mm, and an 84 by 48 pixel resolution. It is connected to the MSP430 microcontroller through a simple SPI bus. This LCD has been proven suitable for portable systems by its extensive use in Nokia cell phones.

For the single-sensor miniEC, no external memory is required. On-board flash memory was used for data storage as well as program memory. The 60 kB of main memory could easily hold the program code and data of at least 100 basic tests. To get equivalent storage capacity for a multi-sensor system, higher capacity memory external to the microcontroller is needed. This is achieved with the addition of the 16-Mbit AT45DB161D Atmel serial DataFlash connected to the microcontroller via USART0 on the same SPI bus as the LCD. Atmel DataFlash storage has several advantages over other forms of non-volatile memory. It is fast, inexpensive and low power. It operates from a single 2.5-3.6V power supply with 3 modes of operation; active (7mA supply current), standby (25 μA supply current) and deep standby (9 μA supply current). These modes allow for management of its power consumption which is very important for battery-operated devices such as

the miniEC. In addition to the main memory, it has 2 SRAM data buffers that enables it to receive and write data to memory at the same time. It supports a minimum of 100,000 write/erase cycles and is rated to retain data for 20 years. Furthermore, all of Atmel's DataFlash memory have the same interface and so memory can be deceased or increased using the same design.

One data point takes up 16 bits of memory, so about 1000000 readings can be stored by the 16-Mbit memory. Assuming 1-second intervals between sensor readings on all five channels, the multi-sensor miniEC has the capability to store about 3000 minutes worth of data.

7.1.6 Firmware

We did not need to redesign all of the firmware for the multi-sensor miniEC since the basic miniEC firmware was designed from the beginning to be modular and easily upgraded. Most of the functions were located in one file though and needed to be abstracted into separate files to ease management and debugging tasks. The majority of the additions/modifications were made in the utility functions that handled the LCD display and hardware.

The microcontroller's port assignments were modified to reflect the change in the LCD and addition of dataflash. The LCD module was rewritten for the new graphical LCD. Minor modifications were made to the main function with its finite state machine core to take advantage of the extra display capability of the LCD. In the hardware module, the major revision was to the ADC code, that was modified to sample from 5 channels instead of 1.

The new data storage module still uses the internal flash memory for data storage and it is still adequate at this time. For future needs, the functions of the storage message can be targeted at the dataflash instead, using Atmel's dataflash

code libraries. Since each recording event samples from 5 channels, a basic test file is now 624 bytes. The size of the header data remains the same as the basic model at 24 bytes but the data section now takes up 600 bytes (5*120 bytes). So, in terms of the number of basic test files, about 3000 files can be stored by the multi-sensor miniEC on the dataflash.

7.2 Evaluation

7.2.1 Materials and methods

General laboratory chemicals and buffer reagents were purchased from VWR Scientific Products (New York, NY, USA), and Sigma Chemical Company. Lipids were obtained from Avanti Polar Lipids (Alabaster, AL, USA). The Dynabeads mRNA Direct Micro Kit was purchased from Dynal Biotech, a division of Invitrogen (Frederick, MD, USA). The NucliSens Basic Kit amplification reagents to perform the nucleic acid sequence based amplification (NASBA) was purchased from BioMrieux (Durham, NC, USA). Oligonucleotides were synthesized by Operon Biotechnologies (Huntsville, AL, USA). Unless specified, all electronic components for the miniEC prototype were purchased from Digikey Corp.(Thief River Falls, MN) and Olimex Ltd (Plodiv, Bulgaria). The Cornell NanoScale Facility provided all the chemicals, wafers, clean room facilities and equipment for microelectrode fabrication.

7.2.2 Microfabrication of IDUAs

The interdigitated ultramicroelectrodes used with the 5 channel miniEC are identical to the basic miniEC model sensors. They were fabricated with standard clean-

room photolithographic and lift-off techniques in a process that varies slightly from methods previously described. The changes primarily allow the IDUAs to be fabricated in a batch process that results in an approximately 6 to 10-fold reduction in production costs. Briefly, a mask was prepared using a pattern generator that was five times the final size. Pyrex wafers (500 μm) backed with a 0.1 μm thick chromium layer were prepared for pattern transfer first by cleaning the top sides in a Hamatech automated hot piranha cleaner and then vapour primed in a YES oven. They were coated with Shipley S-1813 positive photoresist and then baked at 90 C for 90 seconds. The pattern was transferred to the prepared glass wafers using a g-line stepper (GCA Autostep 5X). The wafers were then baked in ammonia gas using a YES image reversal oven to reverse the tone of the photoresist and flood exposed for 60 seconds using an HTG contact aligner. The pattern was revealed after a 1 minute immersion in 300 MIF developer in the Hamatech automated system. Prior to metal deposition, the wafers were submerged in transene chromium etchant to remove the chromium layer and descumed for 20 sec in a UV/Ozone cleaner. A 70 $\mathring{A}$ titanium adhesion layer followed by a 500 $\mathring{A}$ gold layer were deposited using an e-gun source (CVC 4500 Evaporator). The wafers were then soaked overnight in 1165 resist stripper to lift off the excess metal and photoresist leaving the patterned electrode arrays. Before use, the glass wafers were rinsed with deionized water, dried and then diced into single IDUA chips.

7.2.3 Renewal of IDUAs

Repeated use, fouling and oxidation of the surface of the electrode material causes the sensitivity of the IDUAs to decrease over time. IDUAs that exhibited aging in this manner were revivied with a 3-step cleaning procedure to prolong their useful life. First, the IDUAs are placed oxidized under UV light in a JeLight Ozone

Cleaner 144AX with oxygen flow of 0.8 LPM for 10 minutes followed immediately by reduction in a solution of ethyl alcohol for 30 minutes. The IDUAs are rinsed in deionized water and dried for the next step. The second cleaning step is electrochemical stipping via cyclic voltametry in sulphuric acid against an Ag/Agcl reference electrode. Typically, the system is set to cycle between -375 mV and 1,445 mV for at least 8 cycles at 200 mV/s. The final step is sonication in a beaker of iso-propanol at the level 6 setting (VWR Scientific Aquasonic 75D sonicator) for 10 minutes. Before use, the IDUAs are rinsed with deionized water and air dried.

7.2.4 Electrochemical measurements

A serial dilution of ferri/ferrohexacyanide in 0.1 M potassium phosphate buffer, pH 7.0 was prepared with combined concentrations of 0, 0.1, 1, 5, 10, 50 and 100 μM. Five IDUAs, labeled IDUA 1 to IDUA 5, were connected to the corresponding channels on the miniEC with color-coded hooked test leads. The IDUA connected to channel 4 of the miniEC was a non working 'dummy' IDUA. The miniEC was connected to a PC via USB for system power and data collection. In the first set of experiments, a potential of 400 mV was applied across the IDUAs and current generated by 10 μL droplets of each concentration of potassium ferrohexacyanide and potassium ferrihexacyanide (ferro/ferrihexacyanide) on each IDUA were measured. Measurements were taken by the miniEC for a duration of 60 s at 1 s intervals for each combined molarity of 0.1 to 100 μM, in triplicate. Between measurements, electrodes were rinsed with deionized water and dried.

Using the same IDUAs and solutions all measurements were repeated with the Epsilon Electrochemical Workstation (Bioanalytical Systems, West Lafayette, IN), one at a time, for comparison to the miniEC measurements.

A second set of measurements at excitation potentials between 200 to 800mV

were also taken in triplicate. Different concentrations of 10 μL droplets of ferri/ferrocyanide from 0 to 10 μM was used on each IDUA on the 5 corresponding miniEC channels. The arrangement was as follows: 0 μM ferri/ferrocyanide (buffer only) on IDUA 1, channel 1; 1 μM on IDUA 2, channel 2; 5 μM on IDUA 3, channel 3; no IDUA on channel 4, test leads only connected; and 10 μM on IDUA 5, channel 5.

7.2.5 Static detection of specific nucleic acids

RNA derived from nucleic acid sequence based amplification (NASBA) of a segment of *Cryptosporidium parvum* mRNA coding for heat shock was used as a model analyte.

Table 7.1: NASBA primers and DNA capture and reporter probes for amplification and detection of *C. parvum*.

Name	Sequence (5'- 3')
Primer 1	aat tct aat acg act cac tat agg gag aag gta gaa cca cca acc aat aca
Primer 2	aga ttc gaa gaa ctc tgc gct ga
Reporter probe	gtg caa ctt tag ctc cag ttcholesterol
Capture probe	biotinaga ttc gaa gaa ctc tgc gc

The detection protocol was adapted from an optical detection protocol[163] with a lateral flow sandwich assay (LFA) on nitrocellulose strips to allow for static electrochemical detection on IDUAs with the miniEC. The primers and probes used are given in Table 7.1. These have been proven by the LFA assay to be very specific for pathogenic *Cryptosporidium sp.*.

RNA sample preparation

Samples containing 1, 3, 5, or 10 oocysts of C. parvum in 10 μL, counted into tubes by flow cytometry, were obtained from the Wisconsin State Laboratory of Hygiene. Prior to shipping, these samples were heat shocked at 42 °C for 20 min

to induce increased production of heat shock RNA. 100 μL of the lysis/binding buffer from the Dynabeads mRNA Direct Micro Kit were added prior to lysis. This buffer contains 5 mmol L1 dithiothreitol, an RNase inhibitor, which should increase the stability of the target *hsp70* mRNA. The oocysts were then lysed by a freeze–thaw process consisting of five cycles of freezing in an ethanol–dry ice bath and thawing in a 65 °C water bath, with each treatment lasting for 1 min. Samples were then shipped on dry ice. On arrival, all samples were thawed and the mRNA isolated from each sample using the Dynabeads mRNA Direct Micro Kit for isolation of mRNA with oligo-d(T) superparamagnetic beads as per the manufacturers directions.

The target RNA in each sample were then amplified in a NASBA protocol using the NucliSens Basic Kit amplification reagents as per the manufacturers instructions. Briefly, 10 μL of the reagent mix, including 2 pmol per reaction of each of the primers shown in Table 1, were added to the 5 μL samples containing the resuspended oligo-d(T)25 beads and incubated at 65 °C for 5 min and then at 41 C for 5 min. Five microliters of the enzyme mix was then added and the samples were returned to the 41 °C heat block for 5 min. Samples were then transferred to a 41 °C water bath for a 90-min incubation. The NASBA amplicons were stored at -80 C until needed.

Detection assay

5 μg of capture probe coupled magnetic beads, 5 μL of reporter probe coupled liposomes encapsulating 200 mM ferro/ferrihexacyande, 10 μL of hybridization buffer (60% formamide, 6xSSC, 0.8% Ficoll type 400, 0.01% Triton X-100, 0.15 M sucrose) and 5 μL of target RNA (*C. parvum* NASBA amplicons) or nuclease-free water (control) were incubated at room temperature for 15 minutes. The mixture was then placed on a magnetic stand for 2 minutes and the supernatant aspirated.

The beads were washed twice in 50 μL of running buffer (10% formamide, 3xSSC, 0.2% Ficoll type 400, 0.01% Triton X-100, 0.2M sucrose) and resuspended in 25 μL of 60 mM octylglucopyranoside (OG) to lyse any captured liposomes. Liposome lysis releases the encapsulated ferro/ferrihexacyanide into the solution for subsequent detection and quantification. The solution was returned to the magnetic stand for another 2 minutes. 5 μL droplets of the aspirant were analyzed on each of the 5 IDUAs connected to the miniEC in parallel electrochemical measurements at 400 mV for the duration of 1 min with data collected at 1-second intervals.

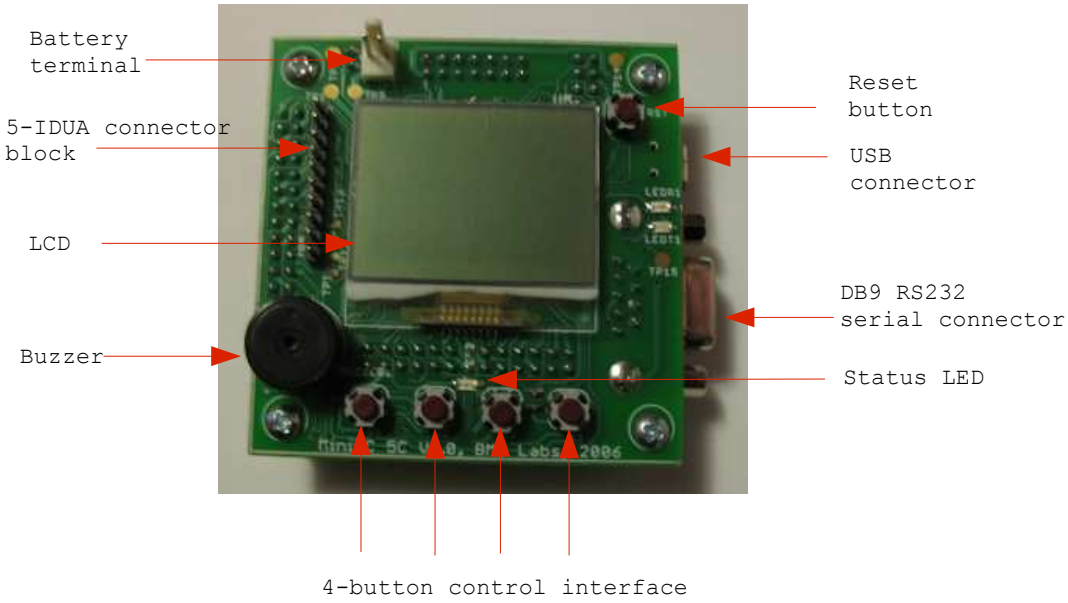
7.3 Results and Discussion

7.3.1 Design overview

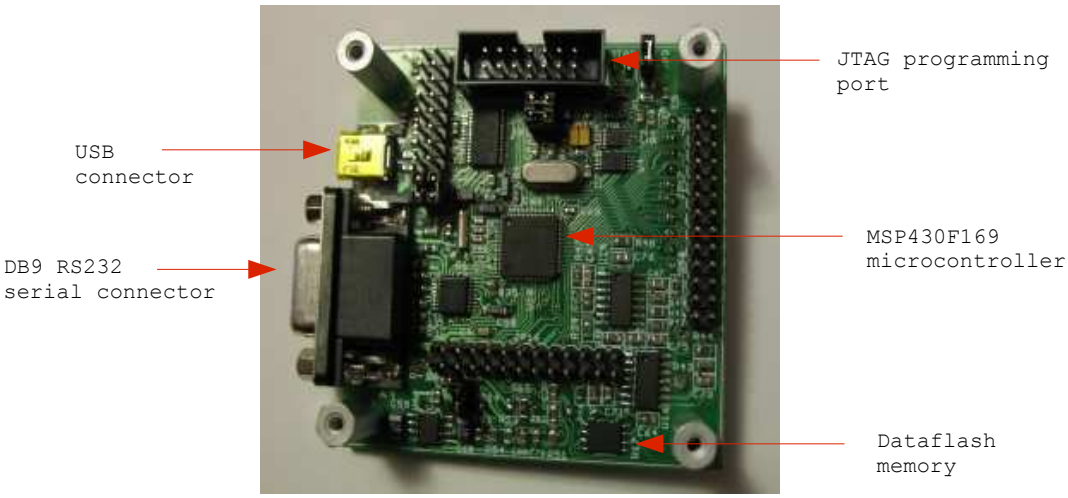
Figure 7.7 shows the top and bottom view of a prototype of the multi-sensor miniEC. It has dimensions of 66 by 66mm and was assembled by Olimex Ltd (Plodiv, Bulgaria). The new miniEC device adds a more sophisticated LCD, extra memory, USB capability and 5-channel sensor control to the base functionality of the single sensor miniEC.

The multi-sensor device also runs off of 2 AAA batteries instead of 1 and can be powered through the USB port. As indicated by the bill of materials in Table 7.2, the difference in terms of component cost between the devices is less than $20, $54 for the multi-sensor and $36 for the basic sensor. The other costs for setup fees and printed circuit board assembly is the same for both units. The multi-sensor miniEC thus represents an excellent value for the price.

All the components selected are rated at or above the conditions for commercial operation at 0 to 45 °C and non-condensing humdity to 95%. The component most likely to limit it's deployment in harsher conditions is the graphical LCD. Since



(a)



(b)

Figure 7.7: Assembled miniEC circuit board showing the positions of some of the major components. The overall board dimensions are 66 by 66 mm.

Table 7.2: Bill of materials for the main components of the 5-channel miniEC

Component	Quantity	Description	Unit price	Cost
TPS60204	1	Charge Pump Converter	$2.36	$2.36
TPS76933	1	Voltage Regulator	$0.96	$0.96
TPS3619-33	1	Battery backup supervisor	$2.48	$2.48
TPS1100	1	FET Switch	$1.41	$1.41
FT232RL	1	USB to RS232 serial chip	$5.00	$5.00
MAX3221	1	RS232 DRIVER	$1.38	$1.38
MAX418	2	Current Feedback op-amp	$4.50	$9.00
MAX1044	1	Negative voltage converter	$2.73	$2.73
AT45DB161D	1	FLASH Memory	$2.58	$2.58
MSP430F169	1	Mixed Signal Microcontroller	$10.00	$10.00
PCD8544	1	Low power LCD	$6.00	$6.00
USB-B-S	1	USB Connector	$0.60	$0.60
DB9	1	sub DB9 female connector	$0.40	$0.40
Other		Resistors, capacitors, etc.		$9.00
		Total Cost		$53.90

the LCD is a standard sized cell-phone LCD, it can easily be replaced with a ruggedized unit typically used for military cell phones.

7.3.2 Electrochemical measurements

Figures 7.8 and 7.9 show comparable signal profiles recorded by the Epsilon and multi-sensor miniEC devices. The general trend was for both sets of profiles to level off after about 20 seconds. The miniEC profiles show more oscillation in the steady state phase especially at the lower concentrations, where the effect of noise is more significant. As the concentration and signal strength increased, this becomes less of a problem. The strength of the miniEC recorded signals was comparable and in most cases higher than those of the BAS Epsilon electrochemical workstation.

The noise pickup however makes the background signal quite high so that the miniEC was less sensitive when the two devices were compared on their signal-

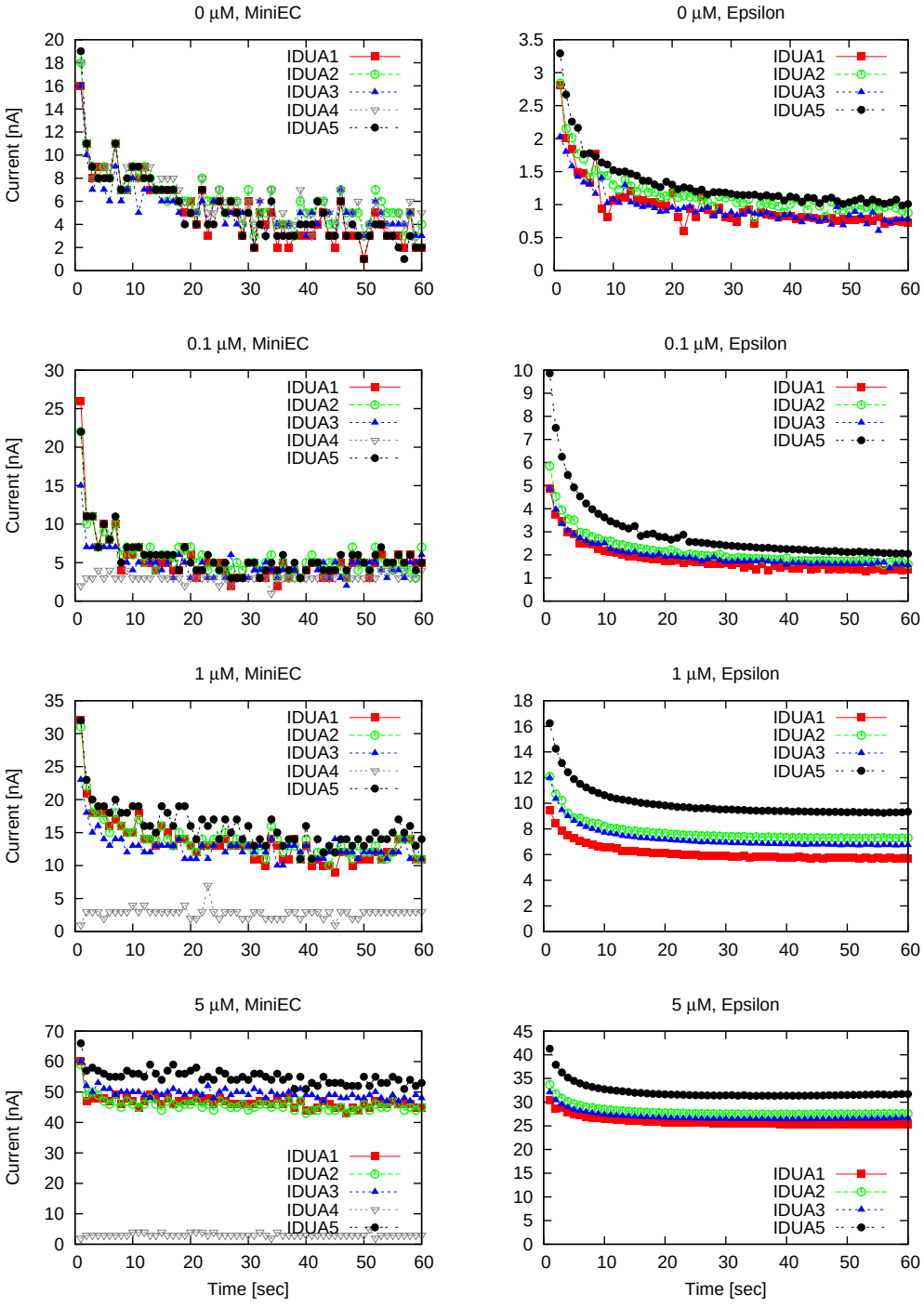


Figure 7.8: Profile of the IDUA current signal measured by the miniEC and Epsilon electrochemical workstation at 1 s intervals for a potential of 400 mV applied for a duration of 60 s. The signals were in response to a 10μL droplet of 0 to 5 μM ferri/ferrohexacyanide placed directly on the IDUAs.

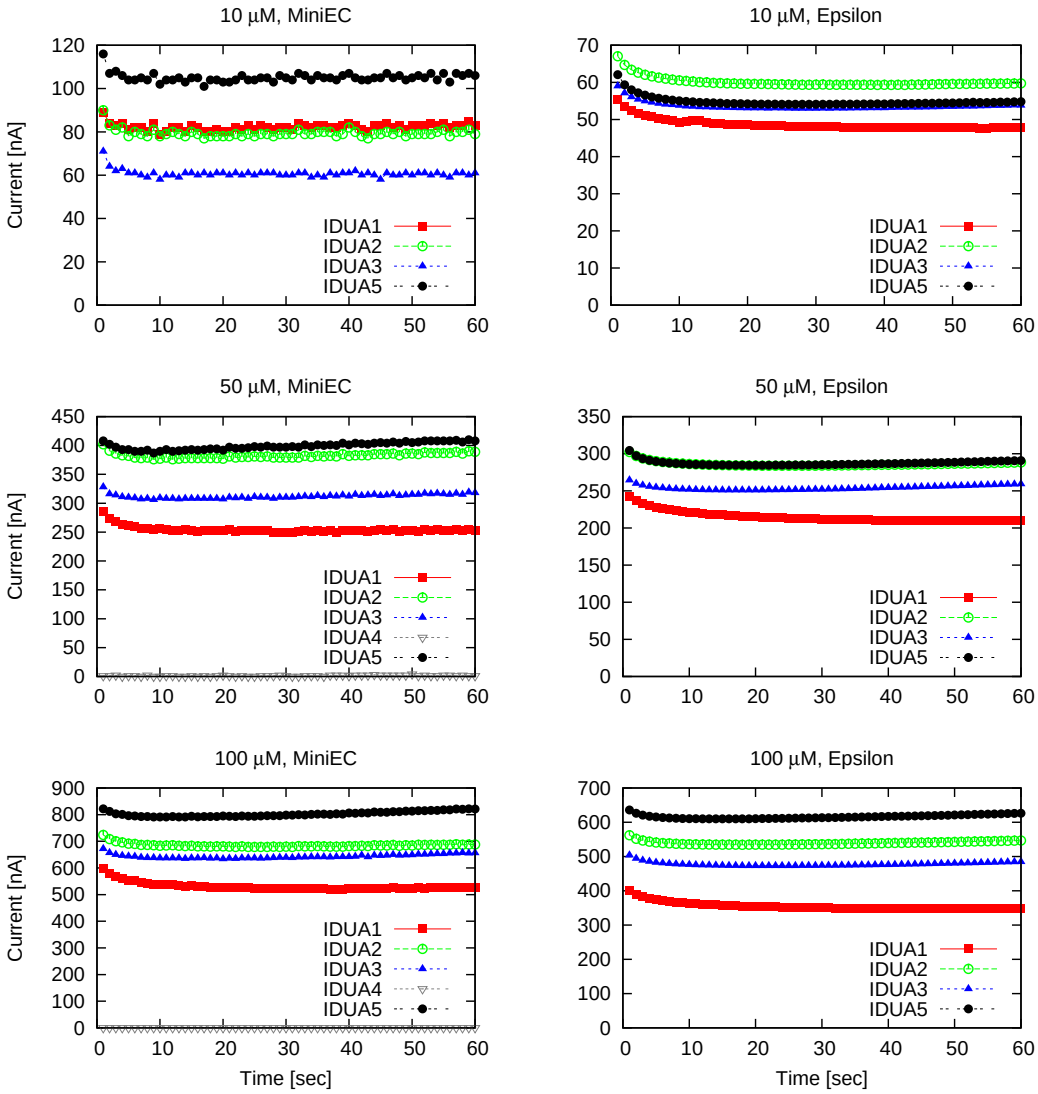


Figure 7.9: Profile of the IDUA current signal measured by the miniEC and Epsilon electrochemical workstation at 1 s intervals for a potential of 400 mV applied for a duration of 60 s. The signals were in response to a 10μL droplet of 10, 50 and 100 μM ferri/ferrohexacyanide placed directly on the IDUA.

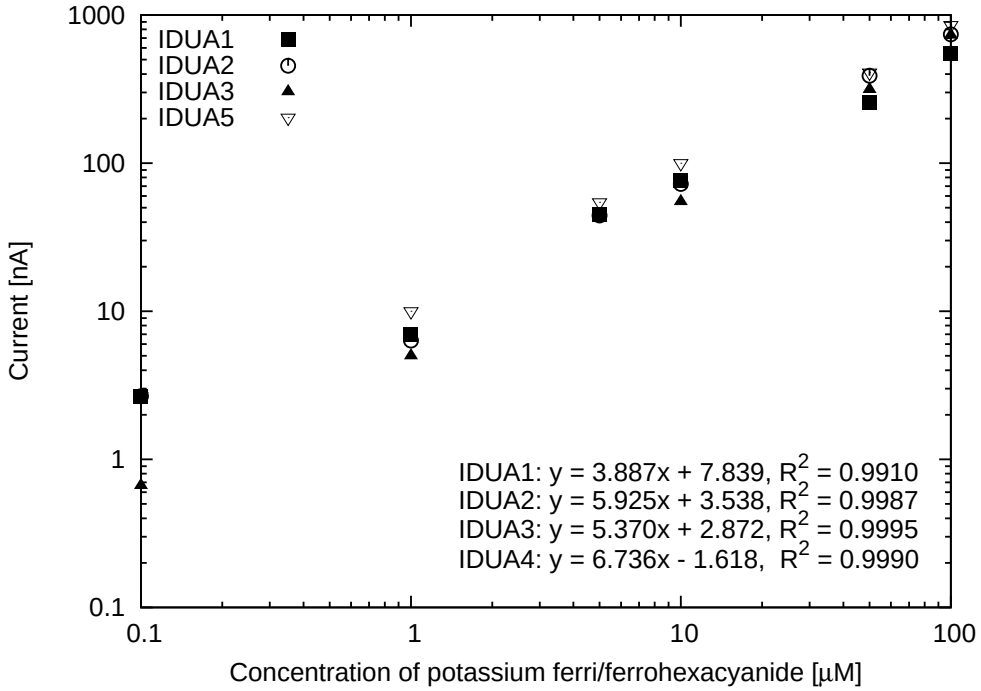


Figure 7.10: Plots showing the concentration curves of four IDUAs for applied potassium ferri/ferrohexacyanide concentrations from 0 to 100 μM.The background signal of 2.67, 4.33, 4.33 and 2.67 nA were subtracted from the datapoints for IDUA 1, 2, 3 and 5 respectively.

Table 7.3: A subset of data for ferri/ferrocyanide detection with IDUA 5, illustrating comparable or higher signals recorded by the miniEC, but lower signal-to-noise ratios calculated due to the miniEC's higher background signal.

Concentration μM	Device	Signal (nA)	Signal-to-noise ratio
0	miniEC	2.67 $\pm$ 0.58	1
	Epsilon	1.05 $\pm$ 0.11	1
0.1	miniEC	5.33 $\pm$ 0.58	2.0
	Epsilon	2.06 $\pm$ 0.05	1.96
1	miniEC	12.67 $\pm$ 1.15	4.75
	Epsilon	10.02 $\pm$ 0.79	9.53
5	miniEC	57.00 $\pm$ 3.61	21.38
	Epsilon	35.33 $\pm$ 3.42	33.58

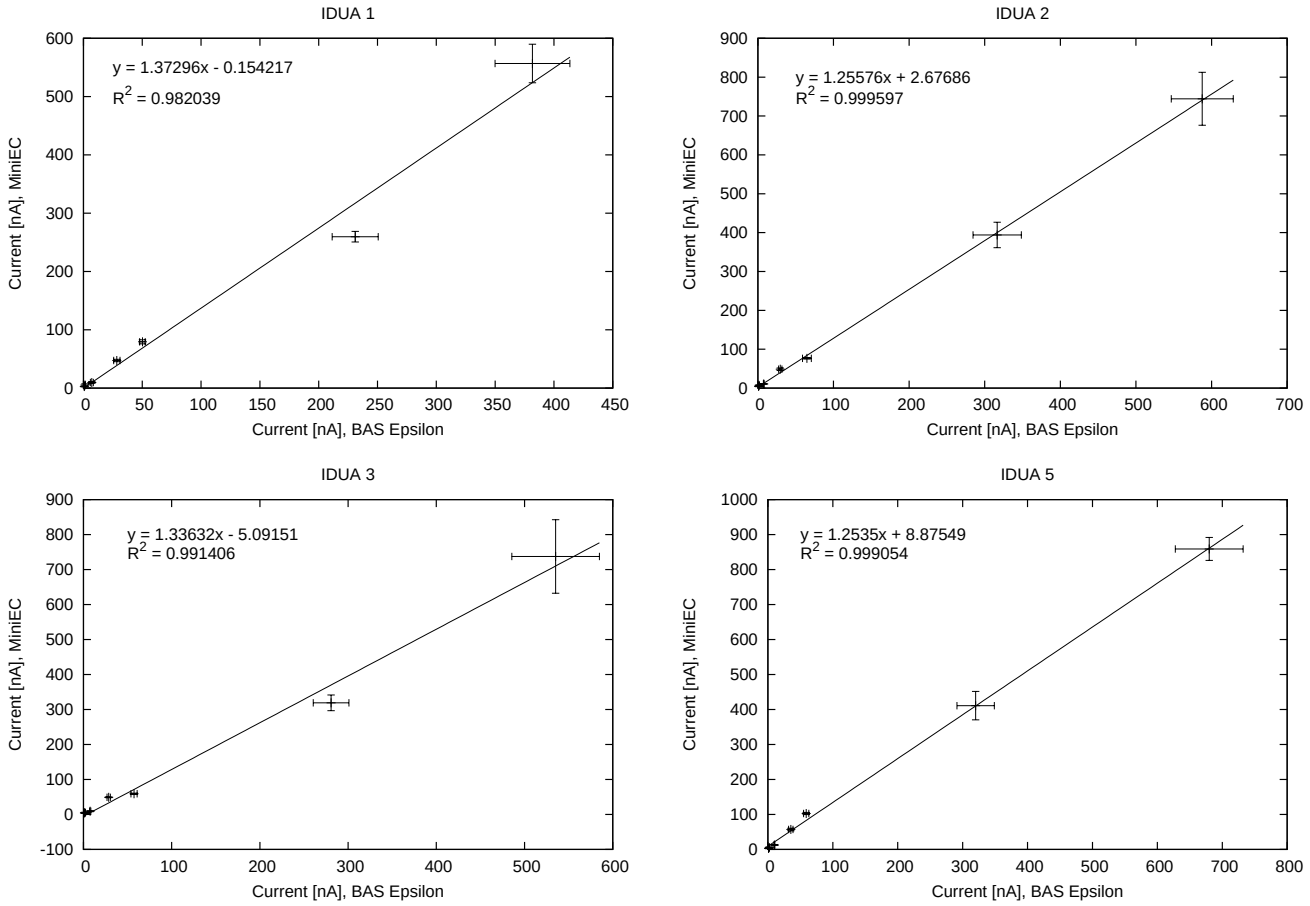


Figure 7.11: Correlation plot comparing the average of three miniEC measurements, to the average of three epsilon measurements of steady state IDUA current at 60s, for combined potassium ferri- and ferrohexacyanide concentrations ranging from 0 to 100 μM. Note the excellent linear correlations $R^2 = 0.982$, 0.999, 0.991 and 0.999 for the 4 IDUAs

to-noise ratios (SNRs). Sensitivity to errant signal conditions is the tradeoff for the combination of very low power consumption of the MSP430 microcontroller and the analog circuitry optimized for small signal detection. Such conditions are more likely in the multi-sensor device with the increase in length and number of test lead cables to accommodate all the sensors. This can be mitigated somewhat by using a suitably designed enclosure to provide some shielding and low noise coaxial or triaxial cables for the electrode connections. Currently, the background on the miniEC has been observed to spike to as high as 7 nA in contrast to about 2 nA for the epsilon under steady state conditions. Consequently, as illustrated by the data in Table 7.3, even when the signal strength was comparable or higher for the miniEC, the SNR for the epsilon was about 2 times higher than it was for the miniEC.

The signals recorded by adjoining channels had no effect on the readings taken by channel 4 of the miniEC which was connected to the the dummy IDUA. In fact, this signal remained random and low, on an order comparable to the background signal throughout the experiments, indicating that cross-talk between the channels is not a problem. The results of measurements with the varying concentrations of ferri/ferrocyanide confirmed this the lack of cross-talk as the signals agreed with the results of the dose-response curve in Figure 7.10.

For both the epsilon and the miniEC, all the signals at every concentration including the background signal (0 μM), increased with increasing potential. This trend was very clear for the epsilon as the background signal doubled for every 200 mV from an average of 0.45 nA at 200 mV to 4.6 nA at 800 mV. The trend was less clear for the miniEC as the increase in the background signal appeared to be within the noise pickup range until the background jumped from an average of 5 nA at 600 mV to 12.33 nA at 800 mV. The potential of 400 mV currently used in

our detection tests had been previously determined experimentally on the epsilon as providing the best combination of signal strength and signal-to-noise ratios. This set of experiments confirmed that 400 mV still has the best combination of signal strength and SNR for the IDUAs on the epsilon. The SNR falls significantly (by almost 50%) from 600mV to 800 mV for both instruments. As a result of the noisy background, the miniEC exhibits a better SNR at 600 mV than at 400 mV.However, measures to reduce the miniEC background rather than increasing the bias potential for miniEC is the preferred solution. This is because higher potentials increase the likelihood of oxidation/reduction of non-target analytes in solution.

7.3.3 Detection of *Cryptosporidium parvum*

Research has begun and is on-going to develop a multi-sensor microfluidic chip that will not only have the ability to detect pathogenic RNA but will also incorporate some of the sample preparation steps such as NASBA to reduce analysis time. The immediate utility and capability of the multi-sensor miniEC system was ably demonstrated, however, with the detection of *Cryptosporidium parvum* in non-stirred 5 μL droplets.

The protocol was adapted from a lateral flow assay (LFA) that was capable of single *C parvum* oocyst detection within 4.5 hours. The LFA assay was a definite improvement over the approved EPA method that can take from 1 to 4 days.[163]

Prior to detection with the miniEC, the samples were all confirmed to be positive with the original lateral flow assay as described by Connelly et al.[163] All sample preparation steps were kept the same between the protocols. The IDUAs in the miniEC system was a direct substitution for the nitrocellulose membranes in the LFA system. Dye encapsulating liposomes tagged with reporter probes were

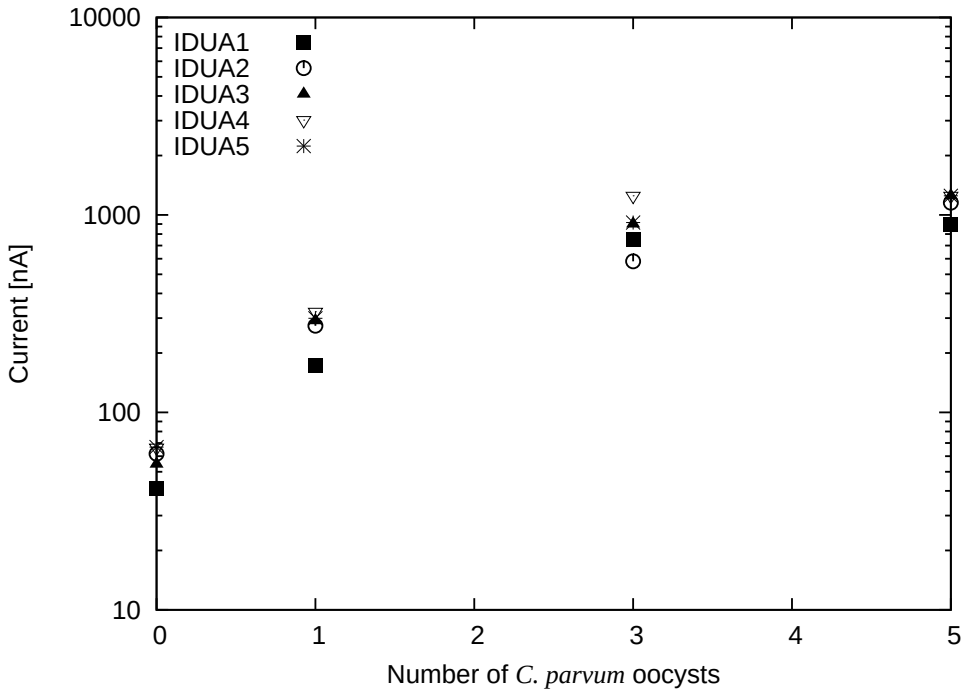


Figure 7.12: Dose-response curve for simultaneous detection of *C. Parvum* amplicons on 5 IDUAs at a constant potential of 400 MV in non-stired 5 μL droplets with the multi-sensor miniEC. Each point is the average for at least 3 replicates. The errors bars were left off as they obscured the data.

replaced with potassium ferri/ferrocyanide encapsulationg ones. The hybridization and running buffer solutions were also changed. The buffer solutions used are the ones that had been optimized and used for the hybridization and electrochemical detection of dengue fever virus as previously described.[133,138] Without the advantages conferred by a microfluidic chip or optimization of hybridization and running buffers, or low noise coaxial cables, the miniEC system was still able to achieve single oocyst detection with good signal-to-noise ratios (Figures 7.12 and 7.13). With the miniEC's transimpedance amplifier gain of 2 MΩ (Figure 7.6), the signal at 5 oocyst was high enough to exceed the maximum measurable current of 1250 nA. As such, the 10-oocyst samples were not tested. The gain can easily be lowered and hence the range extended with the substitution of a smaller resistor in the analog circuitry for a unit dedicated to static non-stirred measurements. The

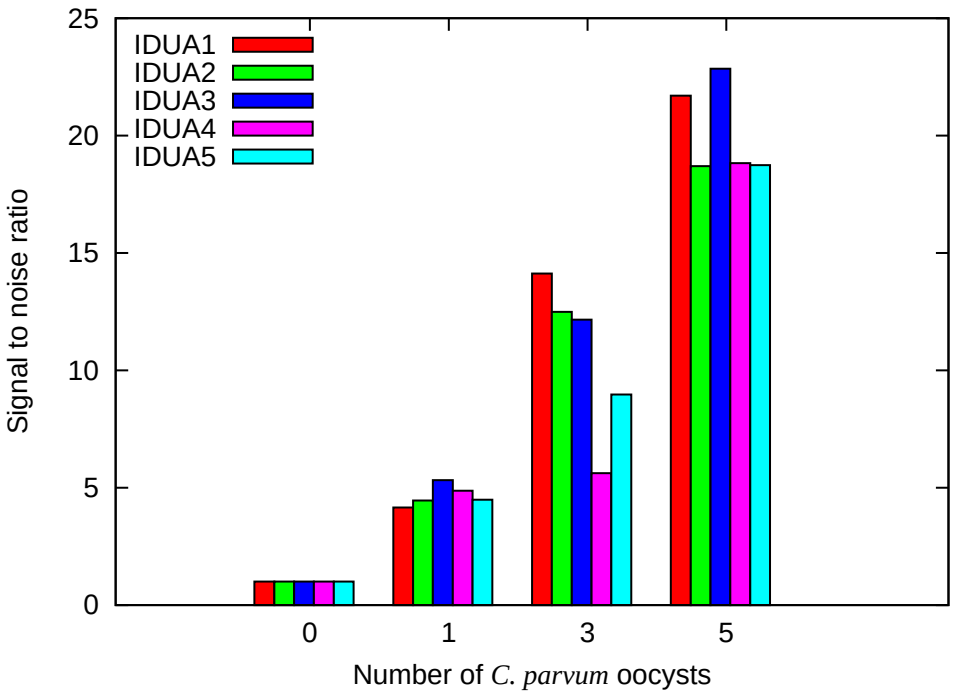


Figure 7.13: Signal-to-noise ratios for the detection of *C. Parvum* amplicons on multi-sensor miniEC.

high gain is more ideal for microfluidic systems where smaller volumes and the constant fluid flow dilutes the analyte resulting in lower signals.

This miniEC detection protocol takes as much time (about 4.5 hours) as the LFA protocol because of the time required for the target preparation steps. The bulk of this time is taken up by incubation during the NASBA amplification step. Both systems are portable and require no more than a magnetic stand and a heating block or water bath. The miniEC system has 2 major advantages that makes it an attractive alternative to the LFA. The first is that handled carefully, the IDUAs can be used, refreshed and reused for hundreds of analyses. The second is the capacity to perform quantitive analyses. With serial dilution of the NASBA amplicons, the miniEC system can be used to quantify the concentration of ferri/ferrocyanide and subsequently, the concentration of the target pathogen in a sample. Instead of serial dilutions to increase the measurement range, however, an

interesting alternative for future research is to investigate whether the same goal can be accomplished with shorter NASBA incubation times.

Conclusion

A miniaturized multi-sensor device was successfully developed using our single-channel miniEC device as a template. The new features include simultaneous amperometric control and measurement from five IDUAs, USB connection for power and data exchange, data storage for up to 3000 basic tests and a larger, easier to read LCD display in a package that is comparable in size and price to the single sensor device. A linear dose-response curve for constant potential amperometric detection of 0 to 100 μM potassium ferri/ferrocyanide was obtained. Furthermore, a correlation of the amperometric detection of potassium ferri/ferrocyanide by the multi-sensor miniEC to that of the commercially available BAS Epsilon electrochemical workstation showed accurate signals with linear correlation coefficients of 0.982, 0.999, 0.991 and 0.999 for four IDUAs. The two systems were also comparable in signal strength, with the miniEC in many cases, exhibiting a higher reading. A higher background on the miniEC however, reduced its comparative signal-to-noise ratio. The multi-sensor device was also successfully used to detect *Cryptosporidium parvum* amplicons derived from a sample containing a single oocyst.

The device can be used in a variety of ways. The same input, for example, can be applied on the all channels providing 5 replicates of one test to speed up analyses or to provide more confidence in a result. Furthermore, with no incidence of cross-channel interference observed, it can also be developed into an inexpensive system to simultaneously quantify and genotype pathogens such as *Cryptosporidium sp.* in the field or multi-serotype pathogens such as Dengue fever virus at point of care locations.

CHAPTER 8

CONCLUSIONS AND FUTURE OUTLOOK

This dissertation has presented a unique three-component framework for miniaturized nucleic acid detection in a handheld package. The first component is a modular hardware platform for ease of application prototyping. As opposed to similar architectures, this system uses commercial-off-the-shelf components selected after extensive research to meet stringent requirements for high performance and low power consumption at low cost.

The second key component is a microfluidic biosensor for rapid, accurate and specific electrochemical detection of pathogens via their RNA. This component encapsulates design knowledge of our lab's best practices within the areas of microfluidic device design and fabrication, nucleic acid hybridization, liposome signal enhancement and electrochemical sensing. To the our knowledge we are the first group to use the powerful combination of signal amplification offered by liposome encapsulation of hundreds of thousands of redox couples with the inherent signal amplification of redox cycling on an ultramicroelectrode array for enhanced sensitivity in a microfluidic biosensor. This component of the platform can be adapted to better reflect the goals of a completely autonomous micro-total analysis system as techniques for miniaturizing sample preparation and target amplification improve.

The final component is a modular and scaleable implementation of the firmware defining the intelligent behavior of the system. This implementation takes into account a number of issues including the real-time reactive nature of the system by using a finite state machine algorithm to run the core of the firmware logic. Because of the diverse fields that include healthcare, food processing and environmental monitoring in which the device can be used and the various ways it

can be customized, the packaging of the device was left as an explicit task for the application designer building on the platform. Some applications such as monitoring standing water for contaminants may be more interested in detecting change rather than a specific quantitative amount of contaminant. Determination of viral load in a clinical sample however may demand precise answers while a test for a condition such as diabetes requires only a yes or no answer.

The basic prototype was evaluated by comparing its performance in a constant potential amperometric detection protocol to that of a commercial electrochemical workstation. An excellent correlation to the commercial system was demonstrated. A multi-sensor prototype was also designed using much of the hardware as the basic prototype. Modifications required to handle the increased throughput were analyzed module by module and component replacements with the same or better performance specifications as the basic system were substituted. Firmware modifications were also straightforward. This exercise provided a guide for application developers interested in customizing the basic platform.

Designing for a mass producible, low cost and user friendly application is usually not a top priority when research scientists are selecting problems to work on. Since the academic community tends to reward the publication of novel ideas over implementation, the focus of much research is usually on making new discoveries or developing deeper understanding of the processes governing a particular technology because it is interesting, or challenging, or both, from a purely scientific point of view. Futhermore, developing an application is time consuming, often costly, with no guarantee that the product will succeed in the marketplace, even if it is superior to comparable devices. One way to reduce the time frame and risks associated with such research is to use already complete units like a laptop or PDA to handle the non-specialized functions such as data storage, display and networking. The gain

in time may be offset by the loss in flexibility to package the product for handheld operation and increased cost. The PDA-based palmsens electrochemical platform mentioned in the introduction, for instance, costs about $4000. The price point of a drug store glucose meter, which is the miniEC's target price, is on the order of $50 for a basic model to about $200 for a top of the line model. The PDA alone in a hybrid system, discounting the cost of customized software required or the cost of the sensor interface, almost certainly exceeds $100.

The mobile phone is another ubiquitous, computing platform that looks very attractive for re-engineering as part of a micro-total analysis system or as simply an input device to display, analyze and transmit data. A basic mobile phone contains a radio antenna connected to a microprocessor that is optimized for processing radio signals. The more sophisticated phones or smart phones, endowed with multiple CPUs and memory are capable of taking pictures, playing music and other tasks such as GPS tracking in addition to the standard phone functions. This fact has not escaped the attention of researchers who have used mobile phones in a variety of sensor applications. Contrary to expectations however, engineering a mobile phone sensor product has turned out to be surprisingly difficult and many of the prototypes that exist are expensive.

One problem these phone–sensor system designers have come up against is a short battery life over which they have no control. In Gay and Leijdekkers's heart monitoring system for instance, the battery life of the mobile phone is 8 hours, while that of the ECG sensor they connect it to, is about 60 hours.[164] The developers rely on the fact that their intended users are likely to be motivated to keep the system charged and operating continuously. In contrast, the miniEC can be operated for over 1000 hours before the batteries need to be replaced. Another problem is that phones with the extra funtionality useful for sensor applications

such as the Audiovox SMT5600, Nokia N95 and the Apple iPhone are unlikely to be the first choice of users in low resource environments given their high retail prices. In some relatively wealthy countries such as the United States, cell phone users rarely pay the full retail price for their phones since the costs are subsidized by phone carriers who make up the difference with 2 to 3 year service contracts. This subsidized market model gives the network carriers veto power over the design of cell phones and their applications. Furthermore, independent developers of sensor-phone devices have no control over the network support required by mobile phone users. Many service providers have limitations that affect how a phone can be used, which may negatively impact the use of a phone-sensor. This problem is discussed by Sascha Segan in an article on the rollout of Motorola's RAZR2 mobile phone in the US. The article makes the point that the RAZR2 is actually 5 different devices with an identical case. The differences are a result of differing hardware and software design decisions made not by Motorola as the manufacturer of the phone, but by the 5 major US mobile phone carriers. Consequently, the RAZR2 phone has different capabilities depending on the service carrier.[165]

The subsidized phone plus subscription model has not worked well in developing countries where the devices will likely be most useful. Mobile phone carriers have instead invested in a prepaid service model with phone cards. The phones while relatively inexpensive compared to the models in more industrialized countries, are still a significant expense for users in these locales. In fact, after the purchase of the phone many users cannot afford the cost of calls. This fact and the problem of dropped calls on fledging networks has led to the development of a new form of communication with a system of intentional missed calls referred to as beeping or flashing.[166] In one form of flashing, participants negotiate a meaning to their missed calls so they can communicate without spending call units. Thus in flash

language, one instumental beep may mean one thing, and two or three beeps may mean another. A phone-sensor device will have to overcome the reluctance of such individuals to use their phone in a way that may add to the cost of ownership. The issue of security especially for sensitive data such as the results of medical tests also needs to be resolved for widespread acceptance of the technology. Other non-technical problems are related to privacy concerns and the potential for embarrassment or prosecution of users whose phones report data from unsavory or restricted locations.

As attractive as the mobile phone is in theory as part of a micrototal analysis system, it has been in practise, hampered by many technological and cultural challenges. For such a system to be feasible, target users of the devices need to have accessible and affordable telecommunication and the developers need open access and freedom to innovate. Until such a time, device platforms like the miniEC are needed to provide portable diagnostic, testing and monitoring services. In addition, further development of the miniEC will advance the technology of portable, inexpensive sensors that in the future can be interfaced to cell phones.

The concept of the miniEC is somewhat similar to that of the Mica mote developed by researchers at University of California Berkeley, in that, it is a generic technology with applications to many different targets. The Mica mote is a generic platform that can be connected to any environmental sensor such as a thermometer, anemometer, or barometer that has a compatible interface. It is now commercially available in a small circuit (2.25 x 1.25 x 0.25 inches rectangular or 1 x 0.25 inches circular) based on an Atmel ATmega 8 bit microcontroller running at about 4 MHz. It has on-board memory and a radio transmitter/receiver for communication. Research is on-going to shrink the mote platform to single chip dimensions. The introduction of the mote revolutionized the field of wireless sens-

ing and brought the concept of "smart dust" closer to reality. It is now the premier experimental platform for researchers developing autonomous sensor networks. In a similar manner, we envision the simple, modular and easily modified technical framework of the miniEC enabling numerous electrochemical microanalytical applications to advance the cause of transferring micro-analysis tools from the lab, into the marketplace.

APPENDIX A

APPENDIX

A.1 Single-sensor MiniEC Firmware Listings

This first iteration of the firmware was made up of two files: fsm5sfg.c and blcda4.c. The blcda4.c file contains the code to control the SBLCDA4 lcd display while the fsm5sfg.c main file contains code for everything else grouped by functionality to implement a finite state machine, for hardware control and data storage.

Listing A.1: Main module for basic miniEC

```
//*****************************************************************************/
//ID:$Id: fsm5sfg.c,v 1.3 2005/10/20 21:25:14 sylvia Exp $

//Author:Sylvia Kwakye
//Date: October 2004
//Compiler: mspgcc
//Target:MSP430FG439
//Version: 1.0
//*****************************************************************************/
#include <signal.h>
#include <io.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "blcda4.c" //lcd code


//hardware for ESG439
#define B1 (1<<6)
#define B2 (1<<7)
#define B3 (1<<1)
#define B4 (1<<2)
#define LED port1.out.pin0
#define LED2 port3.out.pin0
#define LED3 port3.out.pin1
#define LED4 port3.out.pin2
#define TA port2.out.pin0

//constant definitions
#define ON 1
#define OFF 0
#define FTIME 0x0001
#define FIVAL 0x0002
#define FB1 0x0004
```

```
#define FB2 0x0008
#define FB3 0x0010
#define FB4 0x0020
#define FDONE 0x0040
#define FCOM 0x0080

#define CR 0x0D //ASCII code for carriage return
#define LF 0x0A //ASCII code forline feed
#define COL 0x3A //ASCII code for colon, :
#define DASH 0x2D //ASCII code for dash,−
#define SPC 0x20 //ASCII code for space
#define EOT 0x04 //ASCII code for end−of−transmision
#define TAB 0x09 //ASCII code for a horizontal tab

#define RX_MSG_MAX 32 // maximum receive buffer size
#define TX_MSG_MAX 32 // maximum transmit buffer size

#define DATA_START 0x7D00 //start of data memory range
#define DATA_END 0xFDFF //end of data memory range
#define INFO_A 0x1080 //start of segment A in information memory
#define INFO_B 0x1000 //start of segment B in information memory

/******** FSM definitions and variables ***********/
#define NUMBER_EVENTS 8
#define NUMBER_STATES 7

//events
#define NONE 0
#define B1KEY 1
#define B2KEY 2
#define TIMEOUT 3
#define IVAL 4
#define B3KEY 5
#define B4KEY 6
#define COMKEY 7


//states
#define IDLE 0 //0 − display time, Low power mode 3
#define READY 1 //1 − Low power mode 0. Ready... waiting for action
#define RECORD 2 //2 − measurements with storage
#define STOP 3 //3 − stop,
#define SET 4 //4 − user setup
#define LOGS 5 //5 − view summary of files in memory
#define COM 6 //6 − operations directly under PC control


// type definitions
//state table entry
typedef struct {
  void (*Action)(void);
  unsigned char nextState;
} stateEntry;

//time structure
```

```
typedef struct {
  int hours, mins, secs; // time
  int days, month, year; // calendar
}rtcTime;



// state variables
volatile unsigned char eventTray = NONE;
unsigned char mode = IDLE;
unsigned char prevmode = IDLE;


//time variables
unsigned int interval = 1; //interval in seconds
unsigned long ntimes = 1; //interval interval−timer units (s)
unsigned long nticks = 0; //interval counter
unsigned int duration = 5; //duration in minutes
unsigned long dtimes = 300; //duration in interval−timer units (s)
unsigned long dticks = 0; //duration counter
rtcTime startTime; //date and time test starts
rtcTime stopTime; //date and time test ends
rtcTime currentTime = {12,0,0,19,5,2005}; //current date and time
unsigned char initialDate[] = __DATE__ " " __TIME__;
//unsigned char version[]="$Date: 2005/10/20 21:25:14 $, $Revision: 1.3 $";
unsigned char version[] = __DATE__ " " __TIME__;


//measurement variables and flags
static unsigned int potential = 400; //400mV bias potential
//static unsigned int dacValue = 0x0298; // 400mV digital value
static unsigned int dacValue = 0x0298; // 400mV digital value
static long ADCresult; //single channel of ADC readings
static int displayResult = 0; //result to display on LCD
static short fullFlag = OFF; //memory full
static short rxFlag = OFF; //indicates received serial data
static short monitorFlag = OFF;
static char rxBuffer[RX_MSG_MAX]; //USART receive buffer
static char txBuffer[TX_MSG_MAX]; //USART transmit buffer
static unsigned char rxIndex = 0; //pointer to rxBuffer elements
static char gain = 5;


//flash memory
unsigned int *datapointer = (unsigned int *)DATA_START; //start address of data in next file
unsigned char numFiles = 0; //number of files saved in memory
unsigned int totalBytes = (unsigned int)DATA_END − (unsigned int)DATA_START; //memory ↩
    available
//unsigned int bytesUsed = 0; //amount of memory used
unsigned int bytesLeft = (unsigned int)DATA_END − (unsigned int)DATA_START;

//text messages
static unsigned char introMsg[] = {"\n\rMiniEC Sensor Monitor Online ...\n\r"};
static unsigned char helpMsg[] = {"\n\rEC Sensor Commands Menu:\
                                  \r(hp)Help\
```

```
                                    \r(gs)Get Status\
                                    \r(gf)Get Files − data only\
                                    \r(ff)Get Files − time and data\
                                    \r(sc)Clear Files\
                                    \r(sp ival dur pot)Set Parameters\
                                    \r(gt)Get Time\
                                    \r(st day−mon−yr hour:min)Set Time\n\r"};


//prototype functions
//general functions
void doSomething(void);
void stopSomething(void);
void stopIntervalTimer(void);
void startIntervalTimer(int ival, int dur);
void displayRTC(void);
void waitMsec(int ms);
void InitializeSystem(void);
void __inline__ wait(register unsigned int n);

//state related functions
void none(void);
void idle(void);
void ready(void);
void record0(void);
void record(void);
void resume(void);
void monitor(void);
void stop(void);
void set(void);
void logs(void);
void com(void);
void deviceRun(void);
unsigned char mainEventGenerator(void);

//analog and digital conversions
void initDAC(void);
void stopDAC(void);
void initADC(void);
void readADC(void);
void stopADC(void);
void initOA0(void);
void stopOA0(void);

//flash memory and data storage
void initFileSystem(void);
void startDataRecord(void);
void saveResult(void);
void Retrieve(void);
void Erase(void);
void eraseSeg(char seg);
void writeByte (char *address, char value);
void writeWord (int *address, int value);
void writeMsg(char *address, char msg[]);
void makeFileEntry(void);
```

```
void makeDataHeader(void);
void logHeader(int mark1, int mark2);
void logData(int mydata);
void logTail(int marker);
void transmitData(int mydata);
int * findNextLocation(void);


//usart
void initUSART0(void);
void serialSendChar(char c);
void serialSendString(char msg[]);
void serialSendInt(int value);
void serialSendResult(void);
void respondToCmd(void);
void getStatus(void);
void listFiles(void);
void getFiles(void);
void getFormatFiles(void);
void clearFiles(void);
void setParms(void);
void setTime(void);
void getTime(void);
void setDefaults(void);

//macros
#define RX_CMD(a,b) (rxBuffer[0]==a && rxBuffer[1]==b)
#define LEAP(year) (!((year) % 4) && (((year) % 100) || !((year) % 400)))

//interrupt routines
interrupt(BASICTIMER_VECTOR) basic_timer_isr(void);
interrupt(WDT_VECTOR) watchdog_timer_isr(void);
interrupt(PORT1_VECTOR) port1_isr(void);
interrupt(ADC_VECTOR) adc12_isr (void);
interrupt(UART0RX_VECTOR) usart0RX_isr(void);

//main state table
const stateEntry mainStates[NUMBER_EVENTS][NUMBER_STATES] = {
  //noevent
  {{idle,IDLE}, //IDLE
   {none,READY}, //READY
   {none,RECORD}, //RECORD
   {none,STOP}, //STOP
   {none, SET}, //SET
   {none, LOGS}, //LOGS
   {none, COM}}, //COM

  //B1 pressed
  {{idle,IDLE}, //IDLE
   {idle,IDLE}, //READY
   {stop,STOP}, //RECORD
   {idle,IDLE}, //STOP
   {idle, IDLE}, //SET
   {idle, IDLE}, //LOGS
   {idle, IDLE}}, //COM
```

```
//B2 pressed
 {{ready,READY}, //IDLE
  {record0,RECORD}, //READY
  {none,RECORD}, //RECORD
  {ready,READY}, //STOP
  {none, SET}, //SET
  {none, LOGS}, //LOGS
  {none, COM}}, //COM

 //timeout
 {{none,IDLE}, //IDLE
  {none,READY}, //READY
  {stop,STOP}, //RECORD
  {idle,IDLE}, //STOP
  {none, SET}, //SET
  {none, LOGS}, //LOGS
  {none, COM}}, //COM

 //measurement interval
 {{none,IDLE}, //IDLE
  {none,READY}, //READY
  {record,RECORD}, //RECORD
  {none,STOP}, //STOP
  {none, SET}, //SET
  {none, LOGS}, //LOGS
  {none, COM}}, //COM

//B3 pressed
 {{none,SET}, //IDLE
  {none,READY}, //READY
  {none,RECORD}, //RECORD
  {none,STOP}, //STOP
  {logs,LOGS}, //SET
  {com,COM}, //LOGS
  {none,COM}}, //COM

 //B4 pressed
 {{com,COM}, //IDLE
  {none,READY}, //READY
  {none,RECORD}, //RECORD
  {none,STOP}, //STOP
  {idle,IDLE}, //SET
  {set, SET}, //LOGS
  {none,COM}}, //COM

 //pc control connection
 {{com,COM}, //IDLE
  {none,READY}, //READY
  {none,RECORD}, //RECORD
  {com,COM}, //STOP
  {com, COM}, //SET
  {com, COM}, //LOGS
  {com, COM}}, //COM
```

```
}; //end of state transition table


int main(void)
{
  //initialize mcu and peripherals
  InitializeSystem();

  //Enable all interrupts
  _EINT();

 //find the next location to write data
  datapointer = findNextLocation();

  //idle mode
  idle();


  while (1){

    if(rxFlag){respondToCmd();}
    if(fullFlag == OFF){LCDM9 &= 0xF0; LCDM8 &= 0xF1;}
    if(fullFlag){LCDM9 &= 0xFF; LCDM8 &= 0xFE;}
    deviceRun();
    LPM3;
  }
}


/*--------------------------------------
  - fetch an event
  - execute appropriate action
  - perform transition to next state
--------------------------------------*/
void deviceRun(void) {
  // define a local variable
  unsigned char oneEvent;

  //fetch the event..
  oneEvent = mainEventGenerator();

  //execute action..
  mainStates[oneEvent][mode].Action();

  //store current state and transition to next state..
  prevmode = mode;
  mode = mainStates[oneEvent][mode].nextState;
}


/*--------------------------------------
- fetch events in order of priority
- return the highest priority event
--------------------------------------*/
unsigned char mainEventGenerator(void) {
```

```
    //get events in order of priority..

  if(eventTray & FIVAL){
    eventTray &= ~FIVAL;
    return IVAL;
  }

 if(eventTray & FDONE){
    eventTray &= ~FDONE;
    return TIMEOUT;
  }

  if(eventTray & FB1){
    eventTray &= ~FB1;
    return B1KEY;
  }

  if(eventTray & FB2){
    eventTray &= ~FB2;
    return B2KEY;
  }

  if(eventTray & FB3){
    eventTray &= ~FB3;
    return B3KEY;
  }

  if(eventTray & FB4){
    eventTray &= ~FB4;
    return B4KEY;
  }

 if(eventTray & FCOM){
    eventTray &= ~FCOM;
    return COMKEY;
  }

  //default event .
  return NONE;
}


/*****************************************************************************
Event generating interrupts
 - real time clock timer
 - interval timer
 - button presses
*****************************************************************************/

/*-------------------------------------
-Real time clock with basic timer interrupt
-------------------------------------*/
interrupt(BASICTIMER_VECTOR) basic_timer_isr(void){
 if (++currentTime.secs == 60){ //count seconds up to one minute
    currentTime.secs = 0; //reset seconds
```

```
    if (++currentTime.mins == 60){ //count minutes up to one hour
     currentTime. mins = 0; //reset minutes
      if (++currentTime.hours == 24){ //count hours up to one day
    currentTime.hours = 0; //reset hours
    if(++currentTime.days == 32){ //count days up to 1 month
      currentTime.days = 1; //reset days
    }
    else if (currentTime.days == 31){ //for 30 day months
      if ((currentTime.month == 4) || (currentTime.month == 6) ||
          (currentTime.month == 9) || (currentTime.month == 11)){
        currentTime.days=1;
       currentTime.month++;
      }
    }
    //for month = feb and leap year
    else if((currentTime.month == 2) && (currentTime.days == 30)){
      currentTime.days=1;
      currentTime.month++;
    }
    //for month = Feb, not a leap year
    else if ((currentTime.month == 2) && (currentTime.days == 29) &&
        !(LEAP(currentTime.year))){
      currentTime.days=1;
      currentTime.month++;
    }
    if (++currentTime.month == 13){ //count months up to 1 year
      currentTime.month = 1; //reset months and
      currentTime.year++; //count years
    }
      }
    }
  }
 LPM3_EXIT;
}

/*--------------------------------------
- interval timer interrupt
- when nticks == ntimes, put FIVAL in the event tray
- when duration dtimes is reached, put FDONE in the event tray
--------------------------------------*/
interrupt(WDT_VECTOR) watchdog_timer_isr(void){
  if(++nticks == ntimes){
    eventTray |= FIVAL;
    nticks = 0;
  }

  if(++dticks == dtimes){
    eventTray |= FDONE;
    dticks = 0;
  }

  LPM3_EXIT;
}
```

```
/*--------------------------------------
- Port1 interrupt service routine to detect button presses
- FB1 goes into the event tray when button 1 is pressed.
- FB2 goes into the event tray when button 2 is pressed.
- FB3 goes into the event tray when button 3 is pressed.
- FB3 goes into the event tray when button 4 is pressed.
--------------------------------------*/
interrupt(PORT1_VECTOR) port1_isr(void)
{
  waitMsec(10); // debounce switch for 10 ms

  if (P1IFG & B1)
   {
     eventTray |= FB1;
     P1IFG &= ~B1; //clear B1 interrupts flag
   }

  else if (P1IFG & B2)
   {
     eventTray |= FB2;
     P1IFG &= ~B2; // clear B2 interrupts flag
   }


  if (P1IFG & B3)
   {
     eventTray |= FB3;
     P1IFG &= ~B3; //clear B3 interrupts flag
   }

  else if (P1IFG & B4)
   {
     eventTray |= FB4;
     P1IFG &= ~B4; // clear B4 interrupts flag
   }

   LPM3_EXIT;
}


/*****************************************************************************
actions
*****************************************************************************/
/*--------------------------------------
-
--------------------------------------*/
//change nothing
void none(void){
  if(mode == IDLE){
    displayRTC();
  }
}

//low power mode, rtc running
void idle(void){
```

```
  stopSomething();
  displayRTC();
  LED = OFF;
 //lcd_display_num(IDLE);
}

/*-------------------------------------
-
-------------------------------------*/
//ready
void ready(void){

  // determine available memory, how much is needed for data storage
  // with current interval and duration settings and
  // to store the file header and trailer information.
  unsigned int infosize = 36; //size of parameter data
  unsigned int needsize = infosize + (dtimes*8);
  //int i=0;

  displayResult = 0;

  //make sure device is not already running
  stopSomething();

  //initialize peripherals needed
  initDAC();
  initADC();
  //initOA0();

  lcd_display_num(displayResult);
  //lcd_display_num(READY);
  LED |= ON;
  fullFlag = OFF;

  //if not enough memory, notify user but continue
  if(needsize > bytesLeft){
    LCDM9 = 0x01 + 0x02 + 0x04 + 0x08; //display full progress bar
    LCDM8 = 0x02 + 0x04 + 0x08; //display full progress bar
    //lcd_display_num(MEM-FULL);
    fullFlag = ON;
  }

 // debug info
  sprintf(txBuffer, "In READY -- left:%u, file:%d\n",bytesLeft, needsize);
  serialSendString(txBuffer);
  memset(txBuffer, 0, TX_MSG_MAX); //clear txbuffer
}

/*-------------------------------------
- record header, start DAC and recording timer
-changed behaviour to match that of BAS, ie start recording after 1 sec interval
-------------------------------------*/
void record0(void){
  startTime = currentTime;
  logHeader(11111,21111);
```

```
  numFiles++;
  DAC12_0DAT = dacValue; //DAC = 400mV with 2.5V ref
  //doSomething();
  //lcd_display_num(displayResult);
  //lcd_display_num(RECORD);
  // startIntervalTimer(interval,duration-1);

  startIntervalTimer(interval,duration);
  LED =~ LED;
  LPM0;
}

/*-------------------------------------
- record
-------------------------------------*/
void record(void){
  doSomething();
  lcd_display_num(displayResult);
  //lcd_display_num(RECORD);
  LED =~ LED;
}


/*-------------------------------------
- wait a prescribed time
-------------------------------------*/
void stop(void){
  lcd_display_num(displayResult);
  stopIntervalTimer();
  DAC12_0DAT = 0; //DAC = 0mV
  LED |= OFF;
}


/*-------------------------------------
- resume
-------------------------------------*/
//void resume(void){
// unsigned long newDuration = (dtimes - dticks)/4;
// DAC12_0DAT = dacValue; //DAC = 400mV with 2.5V ref
// startTime = currentTime;
// logHeader(41111,21111);
// startIntervalTimer(interval,newDuration);
// LED =~ LED;
//}


/*-------------------------------------
- settings
-------------------------------------*/
void set(void){
 lcd_display_num(SET);
}
```

```
/*--------------------------------------
- view stored data
--------------------------------------*/
void logs(void){
  lcd_display_num(LOGS);
}

/*--------------------------------------
-
--------------------------------------*/
void com(void){
  serialSendString(introMsg);
  respondToCmd();
  lcd_display_num(COM);
}

/*--------------------------------------
//monitor a miniEC experiment
--------------------------------------*/

void monitor(void){
  displayResult = 0;

  //make sure device is not already running
  stopSomething();

  //initialize peripherals needed
  initDAC();
  initADC();
  //initOA0();

  lcd_display_num(displayResult);
  //lcd_display_num(READY);
  LED |= ON;
  fullFlag = OFF;

  //init dac, mode and timer
  DAC12_0DAT = dacValue;
  mode = RECORD;
  startIntervalTimer(interval,duration);

  //blonk led and set monitor flags
  LED =~ LED;
  monitorFlag = ON;

  //enter low power mode 0
  LPM0;
}


//----------------------------------//
//Using the watchdog timer for interval timing of samples
//Sample Intervals = 1 to 60 sec, duration in minutes
//watchdog is clocked by ACLK
//----------------------------------//
```

```
void startIntervalTimer(int ival, int dur){
  nticks = 0;
  ntimes = ival;

  dticks = 0;
  dtimes = dur*60;
  //starttime = hour:mins:sec

  WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer
  IE1 &= ~WDTIE; // disable WDT interrupt
  WDTCTL = WDT_ADLY_1000; // WDT 1s, ACLK, interval mode
  IE1 |= WDTIE; // Enable WDT interrupt
}

void stopIntervalTimer(void){
 WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer
 IE1 &= ~WDTIE; // disable WDT interrupt
}



//delay function
void __inline__ wait(register unsigned int n){
  __asm__ __volatile__ (
              "1: \n"
              " dec %[n] \n"
              " jne 1b \n"
              : [n] "+r"(n)
              );
}

//1ms delay at 8MHz
void waitMsec(int ms){
  volatile int loops = ms;
  while(loops){
    wait(7995);
    loops--;
  }
}

void displayRTC(void){
  displayTime(currentTime.secs,currentTime.mins,currentTime.hours);
}


void doSomething(void){
  readADC();
}

void stopSomething(void){
  stopIntervalTimer();
  stopTime = currentTime;
  stopDAC();
  stopADC();
  monitorFlag = OFF;
```

```
  //stopOA0();
  displayResult = 0;
  //logTail(31111);
  TA = OFF;
  waitMsec(10);
}


/*********************************************************************
analog and digital conversions
*********************************************************************/
/*-----------------------------------

 -----------------------------------*/
/*********************************************************************/

//DAC, initialize and calibrate
void initDAC(void){
  ADC12CTL0 |= REF2_5V + REFON; //2.5V reference on
  DAC12_0CTL |= DAC12OPS|DAC12IR|DAC12AMP_5|DAC12ENC; //VERef+, gain=1
  DAC12_0CTL |= DAC12CALON; //calibrate dac output
  waitMsec(32); //wait for calibration ~32ms
}

void stopDAC(void){
  DAC12_0CTL = 0; // Switch off DAC12
  DAC12_0DAT = 0;
}

//opamps, initialize OA0 in programmable amplifier mode
// with an internal connection to ADC channel 1
void initOA0(void){
  OA0CTL0 = OAPM_3 + OAADC1; // fast power mode,A1 output
  OA0CTL1 = OAFC_4 + OAFBR_2; // +pga mode,gain=1, rail to rail
}

void stopOA0(void){
  OA0CTL0 = OAPM_0; // power mode is off
}


void initADC(void){
  ADC12CTL0 = ADC12ON+MSC+REFON+REF2_5V+SHT0_8; // Setup ADC (sht=16 adcclk cycles)
  ADC12CTL1 = SHP + CONSEQ_2; // sampling timer, repeat mode
  ADC12MCTL0 = SREF_1; // start with channel A0
  ADC12CTL0 |= ENC; // Enable conversions
  ADC12IE = 0x02; // Enable ADC12IFG.1
  wait(33); // 33 cycles delay needed for 8MHz
}

//ADC, stop
void stopADC(void){
  ADC12CTL0 &= ~ENC;
  ADC12CTL0 &= ~ADC12CTL0;
}
```

```
interrupt(ADC_VECTOR) adc12_isr (void){
  volatile int numToAvg = 256;
  unsigned long ADCsum = 0;

  //accumulate 256 readings
  while (numToAvg > 0){
    ADCsum += ADC12MEM0; // Move results, IFG is cleared
    numToAvg--;
  }

    ADCresult = ADCsum >> 8; //divide by 256
    ADCsum = 0; //reset

  //reset
  numToAvg = 256;

  LPM0_EXIT;
}

//ADC read
//For adc measurements using oversampling and averaging methods for noise
//suppression. ADC value then stored/displayed/tramsmited via usart
void readADC(void){
  //initADC();
  ADC12CTL0 |= ADC12SC; // Start ADC sampling and conversion
  LPM0; // wait for conversions to finish

 //Channel data
  ADCresult *= 2500; //multiply by voltage reference
  ADCresult = ADCresult >> 12; //scale by 12 bits
  ADCresult = ADCresult/gain; //scale by gain
  displayResult = (int)ADCresult;

  if(monitorFlag == ON){
    serialSendChar(' ');
    serialSendInt(displayResult);
    serialSendString("\r");
  }

  else{
  logData(displayResult);
  }

}


/*********************************************************************
File data storage (flash timing generator timing = 257 to 476kHz)
*********************************************************************/
/*-----------------------------------

 -----------------------------------*/
```

```
//assemble data file header and write it to file
void logHeader(int mark1, int mark2){
  int filemark = mark1;
  int headmark = mark2;

    writeWord(datapointer, filemark);// write result to file
    writeWord(datapointer+1, startTime.secs);
    writeWord(datapointer+2, startTime.mins);
    writeWord(datapointer+3, startTime.hours);
    writeWord(datapointer+4, startTime.days);
    writeWord(datapointer+5, startTime.month);
    writeWord(datapointer+6, startTime.year);
    writeWord(datapointer+7, potential);
    writeWord(datapointer+8, interval);
    writeWord(datapointer+9, duration);
    writeWord(datapointer+10, headmark);
    datapointer = datapointer+11; // point to next data position in file
    //bytesUsed = bytesUsed+22;
    bytesLeft = bytesLeft - 22;
}

 /*-----------------------------------
  -----------------------------------*/
void logData(int mydata){
  //if space available save data to flash
    if(datapointer < (unsigned int *)DATA_END){
      writeWord(datapointer, mydata);// write result to file
      datapointer++; // point to next data position in file
      bytesLeft = bytesLeft - 2;
  }

  //if full keep measuring but display full progress bar. No storage
  if(datapointer == (unsigned int *)DATA_END){
    LCDM9 = 0x01 + 0x02 + 0x04 + 0x08;
    LCDM8 = 0x02 + 0x04 + 0x08;
  }
}

/*-----------------------------------
  -----------------------------------*/
void transmitData(int mydata){
    serialSendChar(' ');
    serialSendInt(mydata);
    serialSendString("\n\r");
}


void logTail(int marker){
  if(datapointer < (unsigned int *)DATA_END){
    writeWord(datapointer, marker);// write result to file
    writeWord(datapointer+1, stopTime.secs);
    writeWord(datapointer+2, stopTime.mins);
    writeWord(datapointer+3, stopTime.hours);
    writeWord(datapointer+4, stopTime.days);
    writeWord(datapointer+5, stopTime.month);
```

```
    writeWord(datapointer+6, stopTime.year);
    datapointer = datapointer+7; // point to next data position in file
    bytesLeft = bytesLeft−14;
  }

}


/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
  −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
int * findNextLocation(void){
  unsigned int *nextptr = (unsigned int *)DATA_START;
  unsigned int *lastptr = (unsigned int *)DATA_END;
  unsigned int counter = 0;
  numFiles = 0;
  totalBytes = (unsigned int)lastptr − (unsigned int)nextptr;

  // while (nextptr < (int *)DATA_END){
   while (nextptr < (unsigned int *)DATA_END){
      if(*nextptr == 11111){numFiles++;}

      if((*nextptr − 0xFFFF) == 0) {
         bytesLeft = bytesLeft − counter*2;
        break;
      }
      nextptr++;
      counter++;
    }

    if(nextptr == lastptr){
      counter++;
      serialSendString("Memory is full\n\r");
      LCDM9 = 0x01 + 0x02 + 0x04 + 0x08; //display full progress bar
      LCDM8 = 0x02 + 0x04 + 0x08; //display full progress bar
      bytesLeft = bytesLeft − counter*2;
    }
    return nextptr;
}



/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
byte, word and block manipulation functions
each byte or word takes 33/Fftg seconds. eg. 33/410k = 80.5us
each block (64b) takes about 3ms to write
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
//Write a byte value to flash at address where
void writeByte (char *address, char value){

  dint(); //disable interrupts.
  FCTL2 = FWKEY | FSSEL1 | 17; //clock source is MCLK, divisor is 17=>462607 Hz write speed
  while(FCTL3 & BUSY); //wait for BUSY flag to clear
  FCTL3 = FWKEY; //clear lock on flash
  FCTL1 = FWKEY | WRT; //enable flash segment write
  *address = value; //write value to flash
```

```
  FCTL1 = FWKEY; //disable flash segment write
  FCTL3 = FWKEY | LOCK; //reset lock to prevent accidental writes
  eint(); // Restore interrupts
  return;
}

//Write an array of bytes to flash starting at address where
void writeString (char *address, char msg[]){
  dint(); //disable interrupts.
  FCTL2 = FWKEY | FSSEL1 | 17; //clock source is MCLK/17=>462607 Hz write speed
  while (*msg != 0)
    {
      while(FCTL3 & BUSY); //wait for BUSY flag to clear
      FCTL3 = FWKEY; //clear lock on flash
      FCTL1 = FWKEY | WRT; //enable flash segment write
      *address = *msg; //write value to flash
      msg++; //increment pointer
    }
  FCTL1 = FWKEY; //disable flash segment write
  FCTL3 = FWKEY | LOCK; //reset lock to prevent accidental writes
  eint(); // Restore interrupts
  return;
}

//Write a word value to flash at address where
//note- words must be located at even addresses
void writeWord (int *address, int value){
  dint(); //disable interrupts.
  FCTL2 = FWKEY | FSSEL1 | 17; //clock source is MCLK/17=>462607 Hz write speed
  while(FCTL3 & BUSY); //wait for BUSY flag to clear
  FCTL3 = FWKEY; //clear Lock on flash
  FCTL1 = FWKEY | WRT; //enable flash segment write
  *address = value; //write value to flash
  FCTL1 = FWKEY; //disable flash segment write
  FCTL3 = FWKEY | LOCK; //reset lock on flash
  eint(); // Restore interrupts
  return;
}

//Write a block (64b) of words to flash starting at address where
void writeBlock (int *address, int values[]){
  char blkcnt = 32;

  dint(); //disable interrupts.
  FCTL2 = FWKEY | FSSEL1 | 17; //clock source is MCLK/17=>462607 Hz write speed
  while(FCTL3 & BUSY); //wait for busy flag to clear
  FCTL3 = FWKEY; //clear Lock on flash
  FCTL1 = FWKEY | WRT | BLKWRT; //enable flash segment write
  while(blkcnt > 0){
    *address = *values; //write value to flash
    while(FCTL3 & WAIT); //wait for WAIT flag to clear
    values++;
    address++; //point to next location
    blkcnt--;
  }
```

```
  FCTL1 = FWKEY; //disable flash segment write
  while(FCTL3 & BUSY); //wait for busy flag to clear
  FCTL3 = FWKEY | LOCK; //reset lock on flash
  eint(); // Restore interrupts
  return;
}




//-------------------------------------
// erase data in one segment
//-------------------------------------
void eraseSeg(char seg)
{
  unsigned int *fp = (unsigned int *)(seg*0x0200)+DATA_START;

  dint(); //disable interrupts.
  FCTL2 = FWKEY | FSSEL1 | 17; //clock source is MCLK, divisor is 17 for 8MHz
  while(FCTL3 & 0x0001); // wait for BUSY to reset
  FCTL3 = FWKEY; // reset the LOCK bit to enable program/erase
  FCTL1 = FWKEY | ERASE; // set single segment erase function
  *fp = 0xFF; // do a dummy write to start erase
  FCTL3 = FWKEY | LOCK; // lock the flash again
  eint(); //enable interrupts.
  return;

}

//-------------------------------------
// erase all data in data memory range from DATA_START to DATA_END
//-------------------------------------

void clearMem(void)
{
  unsigned int *fp = (unsigned int *)DATA_START;

  dint(); //disable interrupts.
  do
    {
      if ((unsigned int)fp & 0x1000)
    {
      LED |= ON;
    }
      else
    LED = OFF;

      FCTL1 = FWKEY + ERASE;
      *fp = 0x00;

      fp += 0x0100;
    } while (fp < (unsigned int *)DATA_END);

  eint(); //enable interrupts.
  return;
```

```
}



/*
Communications
*/

/*-------------------------------------------
-
-------------------------------------------*/
//usart0, 2400 or 9600 baud rate
void initUSART0(void){
  U0CTL = 0x1; //reset
  UTCTL0 = SSEL0; // UCLK = ACLK

  //UBR00 = 0x0D; // 32k/2400 = 13.65
  //UMCTL0 = 0x6B; // Modulation for 2400

   UBR00 = 0x03; // 9600 from 32768Hz
   UMCTL0 = 0x4A; // Modulation for 9600

  UBR10 = 0x00;
  UCTL0 = CHAR; // 8-bit character *SWRST*
  ME1 |= UTXE0 + URXE0; // Enable USART0 TXD/RXD
  IE1 |= URXIE0; // Enable USART0 RX interrupt
}

/*-------------------------------------------
-
-------------------------------------------*/
//Interrupt to recieve serial data
interrupt(UART0RX_VECTOR) usart0RX_isr(void){
  while ((IFG1 & UTXIFG0) == 0);
  rxBuffer[rxIndex] = RXBUF0; // RXBUF1 to array
  rxIndex++;

  // check for carriage return character or end of buffer
 if ((rxIndex == RX_MSG_MAX) || (rxBuffer[rxIndex-1] == CR))
   {
     IE1 &= ~URXIE0; // disable USART1 RX interrupt
     rxFlag = ON; // turn RX flag on
   }
  LPM3_EXIT;
}

/*-------------------------------------------
-
-------------------------------------------*/
//Transmit 1 byte
void serialSendChar (char c)
{
  while (!(IFG1 & UTXIFG0)); // USART1 TX buffer ready
  TXBUF0 = c;
```

```
}

/*-------------------------------------
-
-------------------------------------*/
//Transmit messages through usart0
void serialSendString (char msg[])
{
  while (*msg != 0)
  {
     while ((IFG1 & UTXIFG0) == 0); //USART TX buffer ready?
     TXBUF0 = *msg; //send character to TXBUF
     msg++; //increment pointer
  }

}

/*-------------------------------------
-
-------------------------------------*/
//send numbers
void serialSendInt(int value){
  char buffer[6];

  if(value == 0){
    serialSendChar('0');
  }
  else {
    sprintf(buffer, "%d", value);
    serialSendString(buffer);
  }
}


/*-------------------------------------
-
-------------------------------------*/
//Respond to commands recieved from usart0
void respondToCmd(void){
  //echo command
  serialSendChar(rxBuffer[0]);
  serialSendChar(rxBuffer[1]);
  serialSendChar(rxBuffer[2]);

  //display help message
  if(RX_CMD('h','p')){serialSendString(helpMsg);}

  //send active signal to pc
  else if(RX_CMD('p','c')){serialSendString("MECOn\n");}

 // PC control
  //else if(RX_CMD('p','c')){eventTray |= FCOM;}

  //start a miniEC experiment
  else if(RX_CMD('m','r')){monitor();}
```

```
  //stop a miniEC experiment
  else if(RX_CMD('m','s')){stopSomething();serialSendString("Done\n");}

  // display current settings
  else if(RX_CMD('g','s')){getStatus();}

  //display files
  else if(RX_CMD('g','f')){getFiles();}
  else if(RX_CMD('f','f')){getFormatFiles();}

  //delete stored files
  else if(RX_CMD('s','c')){clearFiles();}

  //update amperometry parameters
  else if(RX_CMD('s','p')){setParms();}

  //get time
  else if(RX_CMD('g','t')){getTime();}

  //update time
  else if(RX_CMD('s','t')){setTime();}

  //default:display help message
  else{
    serialSendString("Command not valid. Type hp for menu\n");
    //serialSendString(helpMsg);
    }

  //reset buffers and enable rx interrupt
  memset (rxBuffer, 0, RX_MSG_MAX);
  memset (txBuffer, 0, TX_MSG_MAX);
  rxFlag = OFF;
  rxIndex = 0;
  IE1 |= URXIE0;
}

/*-------------------------------------
-
-------------------------------------*/
void getStatus(void){

  //firmware version, current protocol, date,
  serialSendString("EC Monitor version ");
  serialSendString(version);
  serialSendChar('\n');

  serialSendString(" Current Settings\n");
  serialSendString("-----------\n");
  sprintf(txBuffer, "ELECTRODE POTENTIAL(mV)= %d\n", potential);
  serialSendString(txBuffer);
  sprintf(txBuffer, "SAMPLING DURATION(min) = %d\n", duration);
  serialSendString(txBuffer);
  sprintf(txBuffer, "SAMPLING INTERVAL(sec) = %d\n", interval);
  serialSendString(txBuffer);
```

```
  sprintf(txBuffer, "OUTPUT RANGE(nA) = 0 - 1000\n");
  serialSendString(txBuffer);

  serialSendString(" Data Summary\n");
  serialSendString("--------\n");
  sprintf (txBuffer, "Files in memory: %d\n", numFiles);
  serialSendString(txBuffer);
  sprintf (txBuffer, "Total Memory (bytes): %u\n", totalBytes);
  serialSendString(txBuffer);
  sprintf (txBuffer, "Memory left (bytes): %u\n", bytesLeft);
  serialSendString(txBuffer);

  //sprintf(txBuffer, "\n\rdatapointer %d %p %u\n\r",(int)datapointer ,&datapointer,*datapointer);
  //serialSendString(txBuffer);
}


/*--------------------------------------
- get data in all files
--------------------------------------*/
void getFiles(void){
  int *dataptr = (int *)DATA_START; //initialize pointer to first address in data memory

if((unsigned int)datapointer == DATA_START){
    serialSendString("No files in memory\n\r");
    return;
  }

  serialSendString("Getting files\n\r");
  serialSendChar(CR);

  while (*dataptr != 0xFFFF && dataptr <= (int *)DATA_END){
    serialSendInt(*dataptr);
    serialSendChar(CR);
    dataptr++;
  }
  serialSendChar('\n');
}



void getFormatFiles(void){
  int *dataptr = (int *)DATA_START; //initialize pointer to first address in data memory
  int fc = 0;
  int tc = 0;
  int ival = interval;

if((unsigned int)datapointer == DATA_START){
    serialSendString("No files in memory\n\r");
    return;
  }

  //serialSendString("Getting files\n");

  while (*dataptr != 0xFFFF && dataptr <= (int *)DATA_END){
```

```
    //format and send header info
   if(*dataptr == 11111){
     sprintf(txBuffer, "Log File: %d\n\r",fc);
     serialSendString(txBuffer);

      //send the date and time
      sprintf(txBuffer, "%d-%d-%d %d:%d:%d\n\r",
           *(dataptr+4),*(dataptr+5),*(dataptr+6),
           *(dataptr+3),*(dataptr+2),*(dataptr+1));
      serialSendString(txBuffer);

      //send test conditions
      sprintf(txBuffer, "POT:%dmV IVAL:%ds DUR:%ds\n\r",
           *(dataptr+7),*(dataptr+8),*(dataptr+9));
      serialSendString(txBuffer);

       //update data pointer, reset file and time interval counters
      ival = *(dataptr+8);
      dataptr = dataptr+11;
      fc++;
      tc = 0;
   }

   //format and send data
   //send data
   sprintf(txBuffer, "%d %d %d %d %d\n\r",
        tc,*dataptr,*(dataptr+1),*(dataptr+2),*(dataptr+3));
   serialSendString(txBuffer);
   dataptr += 4;
   tc += ival;
  }
}


/*---------------------------------
- clear all the files in memory
---------------------------------*/
void clearFiles(void){
  if((unsigned int)datapointer == DATA_START){
    serialSendString("No files in memory\n\r");
    return;
  }

  serialSendString("Clearing logs\n\r");
  clearMem();

  datapointer = (unsigned int *)DATA_START;
  totalBytes = (unsigned int)DATA_END - (unsigned int)DATA_START; //memory available
  bytesLeft = totalBytes;
  //bytesUsed = 0; //amount of memory used
  numFiles = 0;

  serialSendString("Logs cleared\n\r");
  getStatus();
```

```
}


/*--------------------------------------
- set potential,interval and duration
--------------------------------------*/
void setParms(void){
 volatile int i = 0;

 char delims[] = " ";
 char *result = NULL;

 result = strtok( rxBuffer, delims );
 while( result != NULL ) {
 switch (i){
   case 0: sprintf(txBuffer, "\rcommand is %s", result );
        serialSendString(txBuffer);
        break;

   case 1:
        interval = atoi(result);
        sprintf(txBuffer, "\rinterval (secs) is %s", result );
        serialSendString(txBuffer);
        break;

    case 2:
        duration = atoi(result);
        sprintf(txBuffer, "\rduration (mins) is %s", result );
        serialSendString(txBuffer);
        break;

    case 3:
      potential = atoi(result);
      dacValue = potential * (0xFFFF/2500);
       sprintf(txBuffer, "\rpotential (mV)is %s", result );
       serialSendString(txBuffer);
       break;

    default: break;
     }

     result = strtok( NULL, delims );
     i++;
     }


  sprintf(txBuffer, "i=%d secs, d=%d mins, p=%d mV\n", interval,duration,potential);
  serialSendString(txBuffer);
}


/*--------------------------------------
- set miniEC date
--------------------------------------*/
void setDate(void){
```

```
  serialSendString("Set date function pending\n\r");
}


/*--------------------------------------
- set miniEC time
--------------------------------------*/
void setTime(void){
//stop interrupts
 volatile int i = 0;

 char delims[] = " ";
 char *result = NULL;
 char *param1 = NULL;
 char *param2 = NULL;

 result = strtok( rxBuffer, delims );
 while( result != NULL ) {
 switch (i){
   case 0: sprintf(txBuffer, "\rcommand is \"%s\"", result );
        serialSendString(txBuffer);
        break;

   case 1: param1 = result;
        sprintf(txBuffer, "\rdate is \"%s\"", param1 );
        serialSendString(txBuffer);
        break;

    case 2: param2 = result;
        sprintf(txBuffer, "\rtime is \"%s\"", param2 );
        serialSendString(txBuffer);
        break;

    default: break;
     }

     result = strtok( NULL, delims );
     i++;
     }

}

/*--------------------------------------
- get miniEC time
--------------------------------------*/
void getTime(void){
  sprintf(txBuffer, "\r%d-%d-%d %d:%d:%d\r",
       currentTime.days,currentTime.month,currentTime.year,
       currentTime.hours,currentTime.mins,currentTime.secs);
  serialSendString(txBuffer);
}

/*--------------------------------------
- list summary of the files in memory
--------------------------------------*/
```

```
void listFiles(void){
  serialSendString("List files function pending\n\r");
}

void InitializeSystem(void){
  //System clock setup
  /*
   * Set the system clock to run at 8MHz in active mode
   * An external crystal sets the auxiliary clock ACLK to 32768Hz
   * The master (MCLK) and sub-main clocks (SMCLK) are set by the
   * internally controlled digital clock (DCOCLK) to 8MHz.
   * MCLK = SMCLK = DCO = (n+1) x ACLK
  */

  WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer
  SCFI0 |= FN_4; // x2 DCO frequency, 8MHz nominal DCO
  SCFQCTL = 121; // (121+1) x 32768 x 2 = 7.995 MHz
  FLL_CTL0 = DCOPLUS + XCAP18PF; // DCO+ set so freq = xtal x D x N+1
  wait(65535); // delay 25ms for FLL to lock 1ms~7995 ticks
  wait(65535);
  wait(65535);
  wait(3280);

  //initialize basic timer to real time clock
  IE2 |= BTIE; // enable basic timer interrupt
  BTCTL = BT_ADLY_1000; // set timer to 1 second intervals


  //***** ESG Setup ********
 //port1 setup:P1.0 LED,P1.1,2,6,7 buttons
  P1DIR |= 0x01;
  LED = OFF;
  P1IFG = 0x00; // clear pending interrupts on Port1
  P1IES = B1 + B2 + B3 + B4; // interrupt on falling edge
  P1IE = B1 + B2 + B3 + B4; // enable interrupts on P1.1,2,6,and 7

  //port2 setup:P2.4,5 = USART0 TXD/RXD,P2.0 TA
  P2SEL |= 0x30;
  P2DIR |= 0x11;

  //port3 setup: P3.0,1,.2 = LED2,3,4
  P3DIR |= 0x07;

  //port4 setup: LCD
  P4SEL |= 0x3F;
  P4DIR |= 0xFF;

 //port5: LCD
  //P5SEL = 0xFF;
  P5SEL = 0xFC;
  P5DIR = 0xFF;

  //port6 setup:ADC
  P6SEL |= 0x01; // Enable A/D channel A1
```

```
 P6DIR |= 0xFE;

 //initialize LCD
 lcd_init();
 lcd_clear();

//initialize usart
 initUSART0();
 IE1 |= URXIE0;
}
```

Listing A.2: LCD module for basic miniEC

```
//*****************************************************************************/
// ID: $Id: blcda4.c,v 1.1 2005/04/25 18:03:57 sylvia Exp $
//
//Description: Blcda4 LCD driver for miniEC development board.
//Author:Sylvia Kwakye
//Date: October 2004
//Compiler: mspgcc
//Target:MSP430FG439
//Version: 1.0
//*****************************************************************************/

//**Macros
//Arrows
#define display_AU() LCDM9 |= 0x10;
#define clear_AU() LCDM9 &= 0xEF;
#define display_AR() LCDM9 |= 0x20;
#define clear_AR() LCDM9 &= 0xDF;
#define display_AD() LCDM9 |= 0x40;
#define clear_AD() LCDM9 &= 0xBF;
#define display_AL() LCDM9 |= 0x80;
#define clear_AL() LCDM9 &= 0x7F;

//Colons
#define display_1COL() LCDM5 |= 0x80;
#define clear_1COL() LCDM5 &= 0x7F;
#define display_2COL() LCDM3 |= 0x80;
#define clear_2COL() LCDM3 &= 0x7F;

/*
#define display_1COL() LCDM5 |= 0x08;
#define clear_1COL() LCDM5 &= 0xF7;
#define display_2COL() LCDM3 |= 0x08;
#define clear_2COL() LCDM3 &= 0xF7;
*/

//Decimal points
#define display_1DP() LCDM1 |= 0x80;
#define clear_1DP() LCDM1 &= 0x7F;
#define display_2DP() LCDM2 |= 0x80;
#define clear_2DP() LCDM2 &= 0x7F;
#define display_4DP() LCDM4 |= 0x80;
#define clear_4DP() LCDM4 &= 0x7F;
#define display_6DP() LCDM6 |= 0x80;
#define clear_6DP() LCDM6 &= 0x7F;
#define display_7DP() LCDM7 |= 0x80;
#define clear_7DP() LCDM7 &= 0x7F;

//A0,A1,A2, ANT
#define display_ANT() LCDM10 |= 0x10;
#define clear_ANT() LCDM10 &= 0xEF;
#define display_A2() LCDM10 |= 0x20;
#define clear_A2() LCDM10 &= 0xDF;
#define display_A1() LCDM10 |= 0x40;
#define clear_A1() LCDM10 &= 0xBF;
```

```
#define display_A0() LCDM10 |= 0x80;
#define clear_A0() LCDM10 &= 0x7F;

//Battery status
#define display_BT() LCDM10 |= 0x01;
#define clear_BT() LCDM10 &= 0xFE;
#define display_B1() LCDM10 |= 0x02;
#define clear_B1() LCDM10 &= 0xFD;
#define display_B0() LCDM10 |= 0x04;
#define clear_B0() LCDM10 &= 0xFB;
#define display_BB() LCDM10 |= 0x08;
#define clear_BB() LCDM10 &= 0xF7;

//Progress bar
#define display_PL() LCDM9 |= 0x01;
#define clear_PL() LCDM9 &= 0xFE;
#define display_P0() LCDM9 |= 0x02;
#define clear_P0() LCDM9 &= 0xFD;
#define display_P1() LCDM9 |= 0x04;
#define clear_P1() LCDM9 &= 0xFB;
#define display_P2() LCDM9 |= 0x08;
#define clear_P2() LCDM9 &= 0xF7;
#define display_P3() LCDM8 |= 0x08;
#define clear_P3() LCDM8 &= 0xF7;
#define display_P4() LCDM8 |= 0x04;
#define clear_P4() LCDM8 &= 0xFB;
#define display_PR() LCDM8 |= 0x02;
#define clear_PR() LCDM8 &= 0xFD;

//F1 to F5
#define display_F5() LCDM8 |= 0x01;
#define clear_F5() LCDM8 &= 0xFE;
#define display_F4() LCDM8 |= 0x80;
#define clear_F4() LCDM8 &= 0x7F;
#define display_F3() LCDM8 |= 0x40;
#define clear_F3() LCDM8 &= 0xBF;
#define display_F2() LCDM8 |= 0x20;
#define clear_F2() LCDM8 &= 0xDF;
#define display_F1() LCDM8 |= 0x10;
#define clear_F1() LCDM8 &= 0xEF;


//Signs
#define display_8BC() LCDM11 |= 0x08;
#define clear_8BC() LCDM11 &= 0xF7;
#define display_RX() LCDM11 |= 0x04;
#define clear_RX() LCDM11 &= 0xFB;
#define display_TX() LCDM11 |= 0x02;
#define clear_TX() LCDM11 &= 0xFD;
#define display_ENV() LCDM11 |= 0x01;
#define clear_ENV() LCDM11 &= 0xFE;
#define display_DOL() LCDM11 |= 0x10;
#define clear_DOL() LCDM11 &= 0xEF;
#define display_ERR() LCDM11 |= 0x20;
#define clear_ERR() LCDM11 &= 0xDF;
```

```
#define display_minus() LCDM11 |= 0x40;
#define clear_minus() LCDM11 &= 0xBF;
#define display_MEM() LCDM11 |= 0x80;
#define clear_MEM() LCDM11 &= 0x7F;


//bit values for digit segments
#define d 0x08
#define c 0x04
#define b 0x02
#define a 0x01
#define e 0x40
#define g 0x20
#define f 0x10

static char *LCD = LCDMEM;
const char lcdfont[] = {
  a|b|c|d|e|f, //"0"
  b|c, //"1"
  a|b|d|e|g, //"2"
  a|b|c|d|g, //"3"
  b|c|f|g, //"4"
  a|c|d|f|g, //"5"
  a|c|d|e|f|g, //"6"
  a|b|c, //"7"
  a|b|c|d|e|f|g, //"8"
  a|b|c|d|f|g, //"9"
  0x00, //" "
};

#undef a
#undef b
#undef c
#undef d
#undef e
#undef f
#undef g
#undef h



//prototype functions
void lcd_init (void);
void lcd_clear(void);
void Lcd_display_num(unsigned int);
void displayValue(int value, int stop);
void displayTime(char s,char m,char h);


//Intialize LCD control register
void lcd_init (void)
{
  //Set up basic timer for LCD frequency ACLK/64
  BTCTL |= BT_fLCD_DIV128;
```

```
  // 4mux LCD, segs0−15 = outputs
  LCDCTL |= LCDSG0_3 | LCD4MUX | LCDSON | LCDON;
  lcd_clear();
}



//Set every LCD memory location to 0x00
void lcd_clear(void){
  volatile int i;
  for( i = 0; i < 20; i++ ){
    LCD[i] = 0;
  }
}



void lcd_display_num(int value)
{
  int i;
  int sign = 0;
//static char *LCD = LCDMEM;

  lcd_clear();

  if( value < 0 )
  {
    value = ~value +1;
    sign = 1;
  }
  i = 2;
  while( value > 9 )
  {
    LCD[i] = lcdfont[value%10];
    value = value/10;
    i++;
  }
  LCD[i] = lcdfont[value];

  if(sign)
  {
    display_minus();
  }
}



void displayTime(char s,char m,char h)
{
  char seconds = s;
  char minutes = m;
  char hours = h;

  lcd_clear();
```

```
  LCDM7 = 0;

  LCD[1] = lcdfont[seconds%10];
  LCD[2] = lcdfont[seconds/10];

  LCD[3] |= lcdfont[minutes%10];
  LCD[4] |= lcdfont[minutes/10];

  LCD[5] |= lcdfont[hours%10];
  LCD[6] |= lcdfont[hours/10];

  display_1COL();
  display_2COL();
}
```

A.2 Multi-sensor MiniEC Firmware Listings

The transition of the the system from a single to a multisensor system involved the abstraction of the various parts of the code that had been grouped by function in the first iteration into separate modules. I took advantage of the code review exercise and improved msp430 and mspgcc experience to improve the code where possible. The main module of the multi-sensor implementation miniEC5C.c, includes the various code components with the directive #include module.h instead of #include module.c as is seen in the basic miniEC implementation with the LCD code. This is because, the multi-sensor implementation not only updates the relevant code modules of the basic miniEC, but also compiles the modules into stand-alone units that can be imported into other related or unrelated projects that use similar components via the inclusion of their interface (module.h) files.

A.2.1 Main module for multi-sensor miniEC

Listing A.3: Main module for multi-sensor miniEC

```
#include <io.h>
#include <iomacros.h>
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "fsm.h"
#include "globals.h"
#include "hardware.h"
#include "pcd8544.h"
#include "storage.h"

//constants
#define RX_BUFFER_MAX 32 // usart1 receive buffer size
#define TX_BUFFER_MAX 64 // usart1 transmit buffer size

//variables
unsigned int interval = 1; //interval in seconds
unsigned long ntimes = 1; //interval in timer units (s)
unsigned long nticks = 0; //interval counter
```

```
unsigned int duration = 1; //duration in minutes
unsigned long dtimes = 60; //duration in timer units (s)
unsigned long dticks = 0; //duration counter
unsigned int dsec = 0;
unsigned int dmin = 0;

time_record test_start; //date and time test starts
time_record current_time = {12,0,0,7,1,11}; //current date and time
time_record alarm_time;
unsigned long repeat_time;

unsigned int potential = 400; //400mV bias potential
unsigned int gain = 2; //gain of 2MOhm
unsigned int result[CHANNELS] = {0,0,0,0,0};
unsigned int result_max[CHANNELS] = {0,0,0,0,0};
unsigned long result_avg[CHANNELS] = {0,0,0,0,0};
unsigned long result_sum[CHANNELS] = {0,0,0,0,0};

unsigned char protocol = 0;

unsigned short usart1_rx_flag = OFF; //indicates received serial data
unsigned short monitor_flag = OFF; //indicates presence/absence of pc
unsigned char usart1_rx_index = 0;
unsigned char usart1_tx_buffer[TX_BUFFER_MAX];
unsigned char usart1_rx_buffer[RX_BUFFER_MAX];
const unsigned char introMsg[] = "miniEC "VERSION" "VERSION_DATE" "VERSION_TIME"\r";
const unsigned char helpMsg[] = {"\r\nEC Sensor Commands Menu:\
                                  \r(hp)Help  \
                                  \r(mr)Start experiment \
                                  \r(ms)Stop experiment \
                                  \r(gs)Get Status \
                                  \r(gf)Get Files \
                                  \r(gh)Get file headers \
                                  \r(sc)Clear Files   \
                                  \r(sp ival dur pot)Set Parameters \
                                  \r(gt)Get Time   \
                                  \r(st day mon yr hour min sec)Set Time \
                                  \r(pi)Get product id   \
                                  \r(si)Get serial number   \
                                  \r(vi)Get vendor id\r\n"};

//macros
#define receive_command(a,b) (usart1_rx_buffer[0]==a && usart1_rx_buffer[1]==b)


//function prototypes
void respond_to_cmd(void);
void get_status(void);
void list_files(void);
void get_files(void);
void clear_files(void);
void set_params(void);
void set_time(void);
void get_time(void);
```

```
int main( void ){
/**** INITIALIZATION ****/
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer

  /*
 //set CPU clock to 4MHZ (4096000Hz) using internal DCO and 32768Hz crystal
  BCSCTL1 = 0x00; // select 32768 Hz
  BCSCTL2 = 0x00; // MCLK = SMCLK = DCOCLK
  delay(0xffff); // give osillator some time to settle
  fll_adjust(FLL_MULTIPLIER(4096000, 32768)); // adjust frequency with XT1=32KHz
  */


 //set CPU clock to 8MHZ XT2 high speed crystal
  BCSCTL1 &= ~XT2OFF; // XT2= HF XTAL
  do
    {
      IFG1 &= ~OFIFG; // Clear OSCFault flag
      delay(0xFF); // Time for flag to set
    }
  while ((IFG1 & OFIFG)); // OSCFault flag still set?
  BCSCTL2 |= SELM_2 + SELS; // MCLK= SMCLK= XT2


  //initialize other peripherals
  ports_init();
  lcd_init();
  lcd_set_contrast(0x45);
  delay(0x00ff);
  usart1_init();
  rtc_timerb_setup();

  //initialize variables
  data_pointer = find_next_location();
  FSM_init();

  _EINT(); // enable all interrupts

  lcd_clear();
  lcd_update();
  delay(0xffff);

  while(1){
    if(usart1_rx_flag == ON){respond_to_cmd();}
    FSM_run();
    if(usart1_rx_flag == ON){respond_to_cmd();}
    LPM3;
  }
}


/*--------------------
//Background process: respond to commands recieved from usart1
--------------------*/
```

```
void respond_to_cmd(void){
  //echo command
  //usart1_send_char(usart1_rx_buffer[0]);
  //usart1_send_char(usart1_rx_buffer[1]);
  //usart1_send_char(usart1_rx_buffer[2]);

  //display help message
  if(receive_command('h','p')){usart1_send_string(helpMsg);}

  //display product id
  else if(receive_command('p','i')){
    sprintf(usart1_tx_buffer, "%x\n",PRODUCT_ID);
    usart1_send_string(usart1_tx_buffer);
  }

  //display vendor id
  else if(receive_command('v','i')){
    sprintf(usart1_tx_buffer, "%x\n",VENDOR_ID);
    usart1_send_string(usart1_tx_buffer);
  }

 //display serial id
  else if(receive_command('s','i')){
    sprintf(usart1_tx_buffer, "%s\n",SERIAL_ID);
    usart1_send_string(usart1_tx_buffer);
  }

  // display current settings
  else if(receive_command('g','s')){get_status();}

 //update amperometry parameters
  else if(receive_command('s','p')){set_params();}

  //get time
  else if(receive_command('g','t')){get_time();}

  //update time
  else if(receive_command('s','t')){set_time();}

  //get file headers
  else if(receive_command('g','h'))
    //{usart1_send_string("get files\n");}
    { data_log_summary(); }


  //display files
  else if(receive_command('g','f'))
    //{usart1_send_string("get files\n");}
    { data_log_read(); }

  //delete stored files
  else if(receive_command('s','c'))
    { data_log_erase(); get_status(); }
    //{usart1_send_string("clear files\n");}
```

```
  //start a miniEC experiment
  else if(receive_command('m','r')){
    //Put B2 events in the event tray
    //usart1_send_string("start\n");
    monitor_flag = ON;
    event_tray |= FB2;
    }

  //stop a miniEC experiment
  //put B1 events in the event tray
  else if(receive_command('m','s')){
    usart1_send_string("#stopped\n");
    event_tray |= FB1;
    monitor_flag = OFF;

  }

  //default:display help message
  else{usart1_send_string("#Command not valid. Type hp for menu\n");}

  //reset buffers and enable rx interrupt
  memset(usart1_rx_buffer,0,RX_BUFFER_MAX);
  memset(usart1_tx_buffer,0,TX_BUFFER_MAX);
  usart1_rx_flag = OFF;
  usart1_rx_index = 0;
  usart1_rx_isr_enable();
}


/*--------------------------
-send
- numfiles, totalbytes, bytesleft, pot,dur,interval, range
----------------------------*/

void get_status(void){

  sprintf(usart1_tx_buffer, "[%d-%d-%d,%d:%d:%d,%d,%d,%d,%d,%u,%u,%d,%s,%s]\r\n",
      current_time.day,current_time.month,current_time.year+YEAR1,
      current_time.hour,current_time.min,current_time.sec,
      potential,duration,interval,
      num_files,mem_total,mem_left,
      CHANNELS,SERIAL_ID,VERSION
      );

  usart1_send_string(usart1_tx_buffer);

}

/*--------------------------
-set
- pot,dur,interval
----------------------------*/

void set_params(void){
```

```
  //usart1_send_string("Set Params\n");

  volatile int i = 0;

  char delims[] = " ";
  char *result = NULL;

  result = strtok( usart1_rx_buffer, delims );
  while( result != NULL ) {
    switch (i){
    case 0: //command string
      //do nothing
      break;
    case 1:
      interval = atoi(result);
      break;
    case 2:
      duration = atoi(result);
      break;
    case 3:
      potential = atoi(result);
      break;
    default: break;
     }
     result = strtok( NULL, delims );
     i++;
     }
  get_status();
}


/*----------------------
-send
- time
(st day mon yr hour min sec)
//hour,min,sec,day,mon,yr
----------------------*/

void set_time(void){
  volatile int i = 0;
  char delims[] = " ";
  char *result = NULL;


  result = strtok( usart1_rx_buffer, delims);
  while( result != NULL ) {
    switch (i){
    case 0: //command string
      //do nothing
      break;
    case 1: //date string
      current_time.day = atoi(result);
      break;

    case 2: //date string
```

```
      current_time.month = atoi(result);
      break;


    case 3: //date string
      current_time.year = atoi(result) - YEAR1;
      break;

    case 4: //time string
      current_time.hour = atoi(result);
      break;

    case 5: //time string
      current_time.min = atoi(result);
      break;

    case 6: //time string
      current_time.sec = atoi(result);
      break;

    default: break;
    }

    result = strtok( NULL, delims );
    i++;
  }

  get_time();

}


/*--------------------
-send
- time
//hour,min,sec,day,mon,yr
---------------------*/

void get_time(void){
  //usart1_send_string("Get Time\n");

  sprintf(usart1_tx_buffer, "%2d-%2d-%4d %2d:%2d:%2d\n",
      current_time.day,current_time.month,current_time.year+YEAR1,
      current_time.hour,current_time.min,current_time.sec);

  usart1_send_string(usart1_tx_buffer);
}




/*****************************************************
background interrupts
```

```
**************************************************/
/*----------------------
//Interrupt to recieve serial data
----------------------*/
interrupt (UART1RX_VECTOR) usart1_rx_isr(void)
{
  volatile unsigned char temp;

  // check status register for receive errors
  if (URCTL1 & RXERR)
    {
    // clear error flags by forcing a dummy read
    temp = RXBUF1;
    return;
    }

  while (!(IFG2 & UTXIFG1));
  usart1_rx_buffer[usart1_rx_index] = RXBUF1; // RXBUF1 to array
  usart1_rx_index++;

  // check for carriage return character or end of buffer
  if((usart1_rx_index == RX_BUFFER_MAX) ||
     (usart1_rx_buffer[usart1_rx_index-1] == 0x0D))
    {
     IE2 &= ~URXIE1; // disable USART1 RX interrupt
     usart1_rx_flag = ON; // turn RX flag on
    }
  LPM3_EXIT;

}
```

A.2.2 Interface to FSM module

Listing A.4: FSM module interface

```
#ifndef miniEC_fsm_H
#define miniEC_fsm_H

/*****************************************************************************
 *
 * $Workfile: fsm.c $
 *
 * This module provides the interface to fsm.c, the finite state machine code
 * for the miniEC. It contains structures, definitions and the state tables.
 *
 *
 *
 * No guarantees, warrantees, or promises, implied.
 *
 *
 *NOTES
 *Previous versions of this file had separate transition tables for the
 *sub-states within SETUP and LOGS.The transition into and out of the sub
 *tables was a bit buggy. Combining the tables into main one solved the problem.
 *Subsequent resolution of an unrelated bug suggested that CPU timing may have
 *been the cause of the problem. DCO Clock has since been increased from 1-MHz
 *to 4-MHz. The current code with a single table works well. The substates are
 *indicated by the prefix "s". The transition table is still small enough to
 *be easily maintained and understood. It can be broken up again in the future
 *as the table grows.
 *
 *****************************************************************************/

#include "globals.h"

//constant definitions
#ifndef ON
#define ON 1
#endif
#ifndef OFF
#define OFF 0
#endif

#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif

/*#define B1 (1<<4)
#define B2 (1<<5)
#define B3 (1<<6)
#define B4 (1<<7)*/

#define FTIME 0x0001
```

```
#define FIVAL 0x0002
#define FB1 0x0004
#define FB2 0x0008
#define FB3 0x0010
#define FB4 0x0020
#define FDONE 0x0040

#define NUMBER_EVENTS 7
#define NUMBER_STATES 21

//events
#define NONE 0
#define B1KEY 1
#define B2KEY 2
#define TIMEOUT 3
#define IVAL 4
#define B3KEY 5
#define B4KEY 6

//states
#define IDLE 0 //0 - display time, Low power mode 3
#define READY 1 //1 - Low power mode 0. Ready... waiting for action
#define RECORD 2 //2 - measurements with storage
#define STOP 3 //3 - stop
#define SETUP 4 //4 - user setup
#define LOGS 5 //5 - view summary of files in memory
#define CLOCK 6 //6 - set time and date
#define sINTERVAL 7 //7 - setup interval
#define sDURATION 8 //8 - setup duration
#define sPOTENTIAL 9 //9 - setup potential
#define sRANGE 10 //10 - setup range
#define sPROTOCOL 11 //11 - select a protocol
#define sHOUR 12 //12 - setup time - hour
#define sMIN 13 //13 - setup time - minute
#define sDAY 14 //14 - setup date - day
#define sMON 15 //15 - setup date -month
#define sYEAR 16 //16 - setup date - year
#define sSELECT1 17 //17 - select chioce to view a log
#define sSELECT2 18 //18 - select choice to delete a log
#define sVIEW 19 //19 - view summary of a logged data file
#define sDELETE 20 //20 - delete a logged data file

//variables defined elsewhere
extern unsigned int potential;
extern unsigned int interval;
extern unsigned int duration;
extern unsigned char protocol;
extern unsigned int result[5];
extern unsigned int result_max[5];
extern unsigned long result_avg[5];
extern unsigned long result_sum[5];
extern unsigned char num_files;
extern unsigned int mem_left;
extern unsigned int *data_pointer;
extern unsigned long dtimes;
```

```
extern unsigned long dticks;
extern unsigned int dsec;
extern unsigned int dmin;

// type definitions
//state table entry
typedef struct {
  void (*Action)(void);
  unsigned char state_next;
} state_entry;


//variables defined by fsm.c
extern unsigned char mode;
extern unsigned char mode_prev;
extern volatile unsigned char event_tray;
extern const state_entry states[NUMBER_EVENTS][NUMBER_STATES];

//Prototype functions
void FSM_init(void);
void FSM_run(void);
unsigned char FSM_get_event(void);

// prototype functions for top level actions
void none(void);
void idle(void);
void ready(void);
void record0(void);
void record(void);
void resume(void);
void monitor(void);
void stop(void);
void setup(void);
void logs(void);
void clock(void);

// prototype functions for SETUP state actions
void set_interval(void);
void set_duration(void);
void set_range(void);
void set_potential(void);
void set_protocol(void);
void set_month(void);
void set_day(void);
void set_year(void);
void set_hour(void);
void set_minute(void);

void inc_interval(void);
void inc_duration(void);
void inc_protocol(void);
void inc_potential(void);
void inc_hour(void);
void inc_minute(void);
void inc_month(void);
```

```
void inc_day(void);
void inc_year(void);

void dec_interval(void);
void dec_duration(void);
void dec_potential(void);
void dec_protocol(void);
void dec_hour(void);
void dec_minute(void);
void dec_month(void);
void dec_day(void);
void dec_year(void);

//prototype funtions for the LOGS state actions
void log_view(void);
void log_view_next(void);
void log_view_prev(void);
void log_delete(void);
void log_delete_confirm(void);
void log_delete_abort(void);
void log_select_view(void);
void log_select_delete(void);
void log_summary_display(void);


#endif //miniEC_fsm_H
```

A.2.3 FSM module

Listing A.5: FSM module

```
#ifndef miniEC_fsm_C
#define miniEC_fsm_C
/*****************************************************************************
 *
 * $Workfile: fsm.c $
 *
 * This module provides routines for the finite state machine states, actions,
 * and transitions that concstitutes the miniEC's behaviour.
 *
 * Copyright, Sylvia Kwakye
 * No guarantees, warrantees, or promises, implied.
 *
 *****************************************************************************/

#include <io.h>
#include <iomacros.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#include "hardware.h"
#include "pcd8544.h"
#include "icons.h"
#include "storage.h"
#include "rtc.h"
#include "fsm.h"

//variable definitions
unsigned char mode = IDLE;
unsigned char mode_prev = IDLE;
volatile unsigned char event_tray = NONE;

unsigned int view_file_num = 0;
unsigned int *file_current;
unsigned int *file_prev;


//main state table
const state_entry states[NUMBER_EVENTS][NUMBER_STATES] = {
  //noevent
  {{idle,IDLE}, //IDLE
   {none,READY}, //READY
   {none,RECORD}, //RECORD
   {none,STOP}, //STOP
   {none, SETUP}, //SETUP
   {none, LOGS}, //LOGS
   {none, CLOCK}, //CLOCK
   {none, sINTERVAL}, //setup interval
   {none, sDURATION}, //setup duration
   {none, sPOTENTIAL}, //setup potential
```

```
 {none, sRANGE}, //setup range
 {none, sPROTOCOL}, //setup test file
 {none, sHOUR}, //setup time − hour
 {none, sMIN}, //setup time − minute
 {none, sDAY}, //setup date − day
 {none, sMON}, //setup date − month
 {none, sYEAR}, //setup date − year
 {none, sSELECT1}, //log SELECT1
 {none, sSELECT2}, //log SELECT2
 {none, sVIEW}, //log VIEW
 {none, sDELETE}}, //log DELETE


//B1 pressed
{{idle,IDLE}, //IDLE
 {idle,IDLE}, //READY
 {stop,STOP}, //RECORD
 {idle,IDLE}, //STOP
 {idle, IDLE}, //SETUP
 {idle, IDLE}, //LOGS
 {idle, IDLE}, //CLOCK
 {setup, SETUP}, //interval
 {setup, SETUP}, //duration
 {setup, SETUP}, //potential
 {setup, SETUP}, //range
 {setup, SETUP}, //test
 {clock, SETUP}, //time − hour
 {clock, SETUP}, //time − minute
 {clock, SETUP}, //date − day
 {clock, SETUP}, //date − month
 {clock, SETUP}, //date − year
 {logs, LOGS}, //SELECT1
 {logs, LOGS}, //SELECT2
 {log_select_view, sSELECT1}, //VIEW
 {log_delete_abort, sSELECT2}}, //DELETE


//B2 pressed
 {{ready,READY}, //IDLE
  {record0,RECORD}, //READY
  {none,RECORD}, //RECORD
  {ready,READY}, //STOP
  {set_interval, sINTERVAL}, //SETUP
  {log_select_view, sSELECT1}, //LOGS ****
  {set_hour, sHOUR}, //CLOCK ****
  {set_duration, sDURATION}, //interval
  {set_potential, sPOTENTIAL}, //duration
  {set_range, sRANGE}, //potential
  {set_protocol, sPROTOCOL}, //range
  {setup, SETUP}, //test
  {set_minute, sMIN}, //time − hour
  {set_day, sDAY}, //time − minute
  {set_month, sMON}, //date − day
  {set_year, sYEAR}, //date − month
  {clock, CLOCK}, //date − year
```

```
 {log_view, sVIEW}, //SELECT1
 {log_delete, sDELETE}, //SELECT2
 {none, sVIEW}, //VIEW
 {log_delete_confirm, sSELECT2}}, //DELETE


//duration timeout
{{none,IDLE}, //IDLE
 {idle,IDLE}, //READY
 {stop,STOP}, //RECORD
 {idle,IDLE}, //STOP
 {none, SETUP}, //SETUP
 {none, LOGS}, //LOGS
 {none, CLOCK}, //CLOCK
 {none, sINTERVAL}, //interval
 {none, sDURATION}, //duration
 {none, sPOTENTIAL}, //potential
 {none, sRANGE}, //range
 {none, sPROTOCOL}, //test
 {none, sHOUR}, //time − hour
 {none, sMIN}, //time − minute
 {none, sDAY}, //date − day
 {none, sMON}, //date − month
 {none, sYEAR}, //date − year
 {none, sSELECT1}, //SELECT1
 {none, sSELECT2}, //SELECT2
 {none, sVIEW}, //VIEW
 {none, sDELETE}}, //DELETE


//interval timeout
{{none,IDLE}, //IDLE
 {none,READY}, //READY
 {record,RECORD}, //RECORD
 {none,STOP}, //STOP
 {none, SETUP}, //SETUP
 {none, LOGS}, //LOGS
 {none, CLOCK}, //CLOCK
 {none, sINTERVAL}, //interval
 {none, sDURATION}, //duration
 {none, sPOTENTIAL}, //potential
 {none, sRANGE}, //range
 {none, sPROTOCOL}, //protocol
 {none, sHOUR}, //time − hour
 {none, sMIN}, //time − minute
 {none, sDAY}, //date − day
 {none, sMON}, //date − month
 {none, sYEAR}, //date − year
 {none, sSELECT1}, //SELECT
 {none, sSELECT2}, //SELECT2
 {none, sVIEW}, //VIEW
 {none, sDELETE}}, //DELETE

//B3 pressed
 {{setup,SETUP}, //IDLE
```

```
    {none,READY}, //READY
    {none,RECORD}, //RECORD
    {none,STOP}, //STOP
    {logs,LOGS}, //SETUP
    {clock,CLOCK}, //LOGS
    {idle,IDLE}, //CLOCK
    {inc_interval, sINTERVAL}, //interval
    {inc_duration, sDURATION}, //duration
    {inc_potential, sPOTENTIAL}, //potential
    {none, sRANGE}, //range
    {inc_protocol, sPROTOCOL}, //test
    {inc_hour, sHOUR}, //time − hour
    {inc_minute, sMIN}, //time − minute
    {inc_day, sDAY}, //date − day
    {inc_month, sMON}, //date − month
    {inc_year, sYEAR}, //date − year
    {none, sSELECT1}, //SELECT1
    {log_select_view, sSELECT1}, //SELECT2
    {log_view_next, sVIEW}, //VIEW
    {none, sDELETE}}, //DELETE


   //B4 pressed
   {{clock,CLOCK}, //IDLE
    {none,READY}, //READY
    {none,RECORD}, //RECORD
    {none,STOP}, //STOP
    {idle,IDLE}, //SETUP
    {setup,SETUP}, //LOGS
    {logs,LOGS}, //CLOCK
    {dec_interval, sINTERVAL}, //interval
    {dec_duration, sDURATION}, //duration
    {dec_potential, sPOTENTIAL}, //potential
    {none, sRANGE}, //range
    {dec_protocol, sPROTOCOL}, //test
    {dec_hour, sHOUR}, //time − hour
    {dec_minute, sMIN}, //time − minute
    {dec_day, sDAY}, //date − day
    {dec_month, sMON}, //date − month
    {dec_year, sYEAR}, //date − year
    {log_select_delete, sSELECT2}, //SELECT1
    {none, sSELECT2}, //SELECT2
    {log_view_prev, sVIEW}, //VIEW
    {none, sDELETE}} //DELETE

}; //end of main state transition table


//local prototype functions
interrupt(PORT1_VECTOR) port1_isr(void);

/*************************************************************************
* Function Name : FSM_init
* returns : none
* arguments : none
```

```
* Description : Initializes the miniEC finite state machine
* - no events in the event tray
* - current mode is IDLE
* - previous mode is IDLE
*****************************************************************************/
void FSM_init(void){
  event_tray = NONE;
  mode = IDLE;
  mode_prev = IDLE;
}


/*****************************************************************************
* Function Name : FSM_run
* returns : none
* arguments : none
* Description : Fetches an event, executes appropriate action and performs
                transition to next mode
*****************************************************************************/
void FSM_run(void) {
  unsigned char one_event;

  one_event = FSM_get_event(); //get an event

    states[one_event][mode].Action(); //execute the action
    mode_prev = mode; //record current state
    mode = states[one_event][mode].state_next; //transition to next state
}



/*****************************************************************************
* Function Name :FSM_get_event
* returns : highest priority event
* arguments : none
* Description : fetches events in order of priority from event tray
*****************************************************************************/
unsigned char FSM_get_event(void){

 if(event_tray & FDONE){
    event_tray &= ~FDONE;
    return TIMEOUT;
  }

  if(event_tray & FIVAL){
    event_tray &= ~FIVAL;
    return IVAL;
  }

 //button 1
  if(event_tray & FB1){
    event_tray &= ~FB1;
    return B1KEY;
  }
```

```
 //button 2
  if(event_tray & FB2){
    event_tray &= ~FB2;
    return B2KEY;
  }

 //button 3
  if(event_tray & FB3){
    event_tray &= ~FB3;
    return B3KEY;
  }

 //button 4
  if(event_tray & FB4){
    event_tray &= ~FB4;
    return B4KEY;
  }

  //default event
  return NONE;
}


/*****************************************************************************
* Function Name :port1_isr
* returns : none
* arguments : port 1 interrupt vector – interrupt(PORT1_VECTOR)
* Description : Port1 interrupt service routine to detect button presses.
                Causes exit from low power mode 3 into active mode.
* Notes : FB1 goes into the event tray when button 1 is pressed.
* FB2 goes into the event tray when button 2 is pressed.
* FB3 goes into the event tray when button 3 is pressed.
* FB3 goes into the event tray when button 4 is pressed.  ↩
     *****************************************************************************/
interrupt(PORT1_VECTOR) port1_isr(void){
  delay(1000); // debounce switch

  if (P1IFG & B1)
   {
     event_tray |= FB1;
     P1IFG &= ~B1; //clear B1 interrupts flag
   }

  else if (P1IFG & B2)
   {
     event_tray |= FB2;
     P1IFG &= ~B2; // clear B2 interrupts flag
   }


  if (P1IFG & B3)
   {
     event_tray |= FB3;
     P1IFG &= ~B3; //clear B3 interrupts flag
   }
```

```
  else if (P1IFG & B4)
   {
     event_tray |= FB4;
     P1IFG &= ~B4; // clear B4 interrupts flag
   }

   LPM3_EXIT;
}




/*************************************************************************
* Function Name : none
* returns : none
* arguments : none
* Description : no action
* Notes :
*************************************************************************/
void none(void){}


/*************************************************************************
* Function Name : idle
* returns : none
* arguments : none
* Description : display time and how much memory is left in minutes
* Notes :
*************************************************************************/
void idle(void){

  //port2.out.pin5 = ON;

  unsigned char ampm = 'a';
  unsigned char time_line[14] = "";
  unsigned char date_line[14] = "";
  unsigned char mem_line[14] = "";
  unsigned int mem_time = get_mem_time(interval);
  time_record now = current_time;
  if(now.hour > 12){ampm = 'p'; now.hour -= 12;} //check for am or pm



  sprintf(time_line, "%02d:%02d%c%c", now.hour, now.min, ampm,'m');
  sprintf(date_line, "%2d-%3s-%04d", now.day, month_name[now.month-1],now.year+YEAR1);
  sprintf(mem_line, "%5d Minutes", mem_time);


  lcd_clear();
  lcd_write_string(5,0, "Idle");
  lcd_draw_box(0,0,83,8, PIXEL_XOR);
  lcd_write_string(3,2, time_line);
  lcd_write_string(1,3, date_line);
```

```
  lcd_write_string(0,4, mem_line);
  lcd_update();

  //port2.out.pin5 = OFF;
}


/*****************************************************************************
* Function Name : ready
* returns : none
* arguments : none
* Description : Stop any already running process, check memory
* Notes : Starts an interval timer to time out if not action follows
*****************************************************************************/
void ready(void){

  // port2.out.pin4 = ON;
  //unsigned char buffer[64];

  unsigned int timeout = 5; //5 minutes timeout duration
  unsigned int infosize = sizeof(file_header) + sizeof(file_footer);
  unsigned int mem_needed;

  stop_interval_timer();
  end_measure(); //make sure device is not already running

  // determine available memory and how much is needed for data storage
  // with current interval and duration configuration
  mem_needed = infosize + (((duration*60)/interval)*(sizeof(int)*5));

  memset(result, 0, CHANNELS);
  memset(result_avg, 0, CHANNELS);
  memset(result_sum, 0, CHANNELS);

  lcd_clear(); //clear the LCD
  lcd_write_string(5,0, "Ready");
  lcd_draw_box(0,0,83,8, PIXEL_XOR);


  //if not enough memory, notify user but continue
  if(mem_needed > mem_left){
    //alert_memory(mem_needed);
  lcd_write_string(1,1, "Low memory!");
  lcd_write_string(0,3, "Press -");
  lcd_write_string(0,4, "\"Enter\" to run");
  lcd_write_string(0,5, "\"Stop\" to exit");
  }

  else{
  lcd_write_string(0,2, "Press - ");
  lcd_write_string(0,3, "\"Enter\" to run");
  lcd_write_string(0,4, "\"Stop\" to exit");
  }

  lcd_update();
```

```
  if(monitor_flag == ON){ usart1_send_string("#Ready. Type \'mr\' again to run\r\n");}

  start_interval_timer(interval,timeout);

  // port2.out.pin4 = OFF;
}




/*****************************************************************************
* Function Name : record0
* returns : none
* arguments : none
* Description : First recording activity--starts ADC and DAC, notes time
* and writes file header
* Notes :
*****************************************************************************/
void record0(void){

  //port2.out.pin3 = ON;

  unsigned char line1[14] = "";

  stop_interval_timer();
  ADC_init(); //initialize ADC
  DAC_init(); //initialize DAC

  lcd_clear();
  sprintf(line1, "00:00/%02d:00", duration);
  lcd_write_string(2,0,line1);
  lcd_draw_box(0,0,83,8, PIXEL_XOR);

  lcd_write_string(0, 2,"Current [nA]");
  sprintf(line1, "%4d", result[0]);
  lcd_write_string(4, 4,line1);
  lcd_draw_box(15,28,67,44, PIXEL_OFF);

  lcd_update();

  if(monitor_flag == OFF){
    //at time t=0s, log header, apply bias potential and start interval timer
    time_record now = current_time;
    num_files++;

    //compose header,
    file_header metadata = {(unsigned int)MARK1,
                num_files, potential, interval, duration,
                (now.year+YEAR1), now.month, now.day,
                now.hour, now.min, now.sec,
                (unsigned int)MARK2};

    data_log_header(metadata);
    }
```

```
    else {
    usart1_send_string("#running\r\n");
    }

  DAC_start(potential);
  start_interval_timer(interval,duration);
  LPM0;
  //port2.out.pin3 = OFF;
}



/*****************************************************************************
* Function Name : record
* returns : none
* arguments : none
* Description : take a measurement, display and record data
* Notes :
*****************************************************************************/
void record(void){

  unsigned int i;
  unsigned char line1[14] = "";



 //read ADC. Results stored in global variable result[5]
  memset(result, 0, CHANNELS);
  ADC_read();



  for(i=0;i<CHANNELS;i++){
    //result_sum[i] += result[i];
    if(result[i] > result_max[i]){
      result_max[i] = result[i];
    }
  }

  /*Note:Next block just shows the first value on the screen.
    Switch to full screen when debugging is over by
    uncommenting this block, assign values to each line
    and write results in positions indicated by
    lcd_write_small_string.

    lcd_write_small_string(4,2,"00:00");
    lcd_write_small_string(0,3,"1:0000");
    lcd_write_small_string(8,3,"2:0000");
    lcd_write_small_string(0,4,"3:0000");
    lcd_write_small_string(8,4,"4:0000");
    lcd_write_small_string(0,5,"5:0000");
    lcd_write_string(8,5,"nA");

   */
```

```
  lcd_clear_box(16,29,66,43);
  sprintf(line1, "%4d", result[0]);
  lcd_write_string(4, 4,line1);

  lcd_clear_line(0);
  sprintf(line1, "%02d:%02d/%02d:00", dmin, dsec, duration);
  lcd_write_string(2,0,line1);
  lcd_draw_box(0,0,83,7, PIXEL_XOR);

  lcd_update();
  LED = ~LED;


  if(monitor_flag == ON){data_log_value(result, ON);}
  else{data_log_value(result, OFF);}


}




/*****************************************************************************
* Function Name : stop
* returns : none
* arguments : none
* Description : stops a recording
* Notes : Should be renamed 'stop'
*****************************************************************************/
void stop(void){
  unsigned int timeout = 5; //5 minutes timeout duration
  unsigned char line[10] = "";

  stop_interval_timer();
  DAC_stop();
  ADC_stop();
  delay(0xffff);


  if(monitor_flag == ON){
    usart1_send_string("\n#Done\r\n");
    monitor_flag = OFF;
    mode = STOP;
  }

  /*Note:this just shows the first value on the screen.
    Remember to switch to full screen when debugging is over.
   */
  sprintf(line, "max:%d", result_max[0] );
  lcd_write_small_string(1, 1, line);
  lcd_draw_line(0,8,83,8, PIXEL_ON);
  lcd_update();
```

```
  start_interval_timer(interval,timeout);
}



/*****************************************************************************
* Function Name : setup
* returns : none
* arguments : none
* Description : top of setup menu
* Notes :
*****************************************************************************/
void setup(void){
  lcd_clear();

  display8x8(1,1,up);
  display24x24(0,2,setup24);
  lcd_write_string(5,3, "Setup");
  display8x8(1,5,down);
  lcd_update();
}



/*****************************************************************************
* Function Name : logs
* returns : none
* arguments : none
* Description : top of logs menu
* Notes :
*****************************************************************************/
void logs(void){
  lcd_clear();

  display8x8(1,1,up);
  display24x24(0,2,file24);
  lcd_write_string(5,3, "Logs");
  display8x8(1,5,down);
  lcd_update();

  file_current = data_pointer;
}



/*****************************************************************************
* Function Name : clock
* returns : none
* arguments : none
* Description : top of clock menu
* Notes :
*****************************************************************************/
void clock(void){
  lcd_clear();

  display8x8(1,1,up);
```

```
  display24x24(0,2,clock24);
  lcd_write_string(5,3, "Time");
  display8x8(1,5,down);
  lcd_update();
}


//--------------------//
//interval settings
//--------------------//
/*****************************************************************************
* Function Name : set_interval
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void set_interval(void){
  unsigned char line[14] = "";

  lcd_clear();
  lcd_write_string(2, 0, "DCPA Setup");
  lcd_draw_box(0,0,83,8, PIXEL_XOR);
  lcd_write_string(0,2, "Interval [sec]");

  lcd_clear_line(4);
  sprintf(line, "%2d", interval);
  lcd_write_string(4, 4, line);
  lcd_draw_box(15,28,67,44, PIXEL_OFF);
  lcd_update();
}


/*****************************************************************************
* Function Name : inc_interval
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void inc_interval(void){
  unsigned char line[14] = "";

  if(++interval == 60){interval = 1;}

  lcd_clear_line(4);
  sprintf(line, "%2d", interval);
  lcd_write_string(4, 4, line);
  lcd_update();
}


/*****************************************************************************
* Function Name : dec_interval
* returns : none
```

```
* arguments : none
* Description :
* Notes :
*******************************************************************************/
void dec_interval(void){
  unsigned char line[14] = "";

  if(--interval==0){interval = 59;}

  lcd_clear_line(4);
  sprintf(line, "%2d", interval);
  lcd_write_string(4, 4, line);
  lcd_update();
}


//--------------------//
//duration settings
//--------------------//
/*******************************************************************************
* Function Name : set_duration
* returns : none
* arguments : none
* Description :
* Notes :
*******************************************************************************/
void set_duration(void){
  unsigned char line[14] = "";

  //lcd_clear();

  lcd_write_string(0,2, "Duration [min]");

  sprintf(line, "%2d", duration);
  lcd_clear_line(4);
  lcd_write_string(4, 4,line);
  lcd_draw_box(15,28,67,44, PIXEL_OFF);
  lcd_update();

}


/*******************************************************************************
* Function Name : inc_duration
* returns : none
* arguments : none
* Description :
* Notes :
*******************************************************************************/
void inc_duration(void){
  unsigned char line[14] = "";

  if(++duration == 61){duration = 1;}

  lcd_clear_line(4);
```

```
  sprintf(line, "%2d", duration);
  lcd_write_string(4, 4, line);
  lcd_update();
}


/*****************************************************************************
* Function Name : dec_duration
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void dec_duration(void){
  unsigned char line[14] = "";

 if(--duration == 0){duration = 60;}

  lcd_clear_line(4);
  sprintf(line, "%2d", duration);
  lcd_write_string(4, 4, line);
  lcd_update();
}

//--------------------//
//potential settings
//--------------------//
/*****************************************************************************
* Function Name : set_potential
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void set_potential(void){
  unsigned char line[14] = "";

  lcd_write_string(0, 2,"Potential [mV]");
  sprintf(line, "%4d", potential);
  lcd_clear_line(4);
  lcd_write_string(4, 4,line);
  lcd_draw_box(15,28,67,44, PIXEL_OFF);

  lcd_update();

}


/*****************************************************************************
* Function Name : inc_potential
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
```

```
void inc_potential(void){
  unsigned char line[14] = "";

  if(++potential > 2500){potential = 2500;}

  sprintf(line, "%4d", potential);
  lcd_clear_line(4);
  lcd_write_string(4, 4, line);
  lcd_update();
}


/*****************************************************************************
* Function Name : dec_potential
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void dec_potential(void){
  unsigned char line[14] = "";

  if(--potential == 0){potential = 1;}

  sprintf(line, "%4d", potential);
  lcd_clear_line(4);
  lcd_write_string(4, 4, line);
  lcd_update();
}


//--------------------//
//range settings
//--------------------//
/*****************************************************************************
* Function Name : ready
* returns : none
* arguments : none
* Description :
* Notes : Range is currently fixed in this application
*****************************************************************************/
void set_range(void){

  lcd_clear_line(2);
  lcd_clear_box(15,28,67,44);

  lcd_write_string(0,2, "RANGE [nA]");
  lcd_write_string(3, 4, "0 - 1250");
  lcd_draw_box(10,28,72,44, PIXEL_OFF);
  lcd_update();
}



//--------------------//
```

```
//--- pre-programmed protocols ---//
//--------------------//
/*****************************************************************************
* Function Name : set_protocol
* returns : none
* arguments : none
* Description : select a pre-programmed protocol
* Notes :
*****************************************************************************/
void set_protocol(void){
  unsigned char line[14] = "";

  lcd_clear();
  lcd_write_string(2, 0, "Test Select");
  lcd_draw_box(0,0,83,8, PIXEL_XOR);

  lcd_write_string(0,2,"Protocol");

  sprintf(line, "Test-%d", protocol);
  lcd_write_string(3, 4, line);
  lcd_draw_box(10,28,72,44, PIXEL_OFF);
  lcd_update();
}


/*****************************************************************************
* Function Name : inc_protocol
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void inc_protocol(void){
  unsigned char line[14] = "";

  if(++protocol == 6){protocol = 1;}

  lcd_clear_line(4);
  sprintf(line, "Test-%d", protocol);
  lcd_write_string(3, 4, line);
  lcd_update();
}


/*****************************************************************************
* Function Name : dec_protocol
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void dec_protocol(void){
   unsigned char line[14] = "";

  if(--protocol == 0){protocol = 5;}
```

```
  lcd_clear_line(4);
  sprintf(line, "Test-%d", protocol);
  lcd_write_string(3, 4, line);
  lcd_update();
}


//--------------------//
//hour settings
//--------------------//
/*****************************************************************************
* Function Name :set_hour
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void set_hour(void){
   unsigned char line[14] = "";

  lcd_clear();
  lcd_write_string(5, 0, "Time");
  lcd_draw_box(0,0,83,8, PIXEL_XOR);

  lcd_write_string(0,2, "Set hour");

  //display time and date
  sprintf(line, "%02d : %02d", current_time.hour,current_time.min);
  lcd_write_string(3, 4, line);
  lcd_draw_box(10,28,72,44, PIXEL_OFF);

  //highlight hour
  lcd_draw_box(15,31,33,40, PIXEL_XOR);

  lcd_update();

  //debug

  //sprintf(line, "\rhour %d %d %d", event_tray, mode_prev, mode);
  //usart1_send_string(line);


}

/*****************************************************************************
* Function Name : inc_hour
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void inc_hour(void){
   unsigned char line[14] = "";
```

```
  if(++current_time.hour == 24){current_time.hour = 0;}

  //lcd_clear_line(4);
  //lcd_clear_box(15,31,33,40); //remove highlight
  sprintf(line, "%02d : %02d", current_time.hour,current_time.min);
  lcd_write_string(3, 4, line);
  lcd_draw_box(15,31,33,40, PIXEL_XOR);

    //debug
  //sprintf(line, "\rhour+ %d %d %d", event_tray,mode_prev, mode);
  //usart1_send_string(line);


  lcd_update();
}

/*****************************************************************************
* Function Name : dec_hour
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void dec_hour(void){
   unsigned char line[14] = "";

  if(current_time.hour == 0){current_time.hour = 23;}
  else{current_time.hour--;}

  lcd_clear_line(4);
  lcd_clear_box(15,31,33,40); //remove highlight
  sprintf(line, "%02d : %02d", current_time.hour,current_time.min);
  lcd_write_string(3, 4, line);
  lcd_draw_box(15,31,33,40, PIXEL_XOR);

  //debug
  //sprintf(line, "\rhour- %d %d %d", event_tray,mode_prev, mode);
  //usart1_send_string(line);


  lcd_update();
}

//--------------------//
//minute settings
//--------------------//
/*****************************************************************************
* Function Name : set_minute
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void set_minute(void){
   unsigned char line[14] = "";
```

```
  lcd_write_string(0,2, "Set minute");
  lcd_clear_box(11,29,71,43);

  sprintf(line, "%02d : %02d", current_time.hour,current_time.min);
  lcd_write_string(3, 4, line);
  lcd_draw_box(10,28,72,44, PIXEL_OFF);

  //highlight minute
  lcd_draw_box(44,31,62,40, PIXEL_XOR);

  //debug
  //sprintf(line, "\rmin %d %d %d", event_tray,mode_prev, mode);
  //lcd_write_string(0,1, line);
  //usart1_send_string(line);


  lcd_update();
}


/*****************************************************************************
* Function Name : inc_minute
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void inc_minute(void){
   unsigned char line[14] = "";

  if(++current_time.min == 60){current_time.min = 0;}

  lcd_clear_line(4);
  lcd_clear_box(44,31,62,40); //remove highlight
  sprintf(line, "%02d : %02d", current_time.hour,current_time.min);
  lcd_write_string(3, 4, line);

  //highlight minute
  lcd_draw_box(44,31,62,40, PIXEL_XOR);

  //debug
  //sprintf(line, "\rmin+ %d,%d %d", event_tray,mode_prev, mode);
  //usart1_send_string(line);


  lcd_update();
}


/*****************************************************************************
* Function Name : dec_minute
* returns : none
* arguments : none
* Description :
```

```
* Notes :
*******************************************************************************/
void dec_minute(void){
   unsigned char line[14] = "";

  if(current_time.min == 0){current_time.min = 59;}
  else{current_time.min--;}

  lcd_clear_line(4);
  lcd_clear_box(44,31,62,40); //remove highlight
  sprintf(line, "%02d : %02d", current_time.hour,current_time.min);
  lcd_write_string(3, 4, line);

  //highlight minute
  lcd_draw_box(44,31,62,40, PIXEL_XOR);

  //debug
  //sprintf(line, "\rmin- %d,%d %d", event_tray,mode_prev, mode);
  //usart1_send_string(line);


  lcd_update();
}

//--------------------//
//--- month settings ---//
//--------------------//
/*******************************************************************************
* Function Name : set_month
* returns : none
* arguments : none
* Description :
* Notes :
*******************************************************************************/
void set_month(void){
  unsigned char line[14] = "";

  lcd_clear_line(2);
  lcd_write_string(0,2, "Set month");
  lcd_clear_box(5,29,80,43);

  sprintf(line, "%02d-%3s-%04d", current_time.day, month_name[current_time.month-1],current_time.year ↩
       +YEAR1);
  lcd_write_string(2, 4, line);

  //highlight month
  lcd_draw_box(29,31,47,40, PIXEL_XOR);

  lcd_update();

}

/*******************************************************************************
* Function Name : inc_month
* returns : none
```

```
* arguments : none
* Description :
* Notes :
*******************************************************************************/
void inc_month(void){
  unsigned char line[14] = "";

  if(++current_time.month == 13){current_time.month = 1;}

  lcd_clear_line(4);
  lcd_clear_box(29,31,47,40); //remove highlight

  sprintf(line, "%02d-%3s-%04d", current_time.day,
                                 month_name[current_time.month-1],
                                 current_time.year+YEAR1);
  lcd_write_string(2, 4, line);
  lcd_draw_box(29,31,47,40, PIXEL_XOR);
  lcd_update();
}

/*******************************************************************************
* Function Name : dec_month
* returns : none
* arguments : none
* Description :
* Notes :
*******************************************************************************/
void dec_month(void){
  unsigned char line[14] = "";

  if(current_time.month == 1){current_time.month = 12;}
  else{current_time.month--;}

  lcd_clear_line(4);
  lcd_clear_box(29,31,47,40); //remove highlight

  sprintf(line, "%02d-%3s-%04d", current_time.day,
                                 month_name[current_time.month-1],
                                 current_time.year+YEAR1);
  lcd_write_string(2, 4, line);
  lcd_draw_box(29,31,47,40, PIXEL_XOR);
  lcd_update();
}



//--------------------//
//day settings
//--------------------//
/*******************************************************************************
* Function Name : set_day
* returns : none
* arguments : none
* Description :
* Notes :
```

```
*****************************************************************************/
void set_day(void){
  unsigned char line[14] = "";

  lcd_clear();
  lcd_write_string(2, 0, "Time Setup");
  lcd_draw_box(0,0,83,8, PIXEL_XOR);
  lcd_write_string(0,2, "Set day");

  lcd_clear_line(4);
  sprintf(line, "%02d-%3s-%04d", current_time.day, month_name[current_time.month-1],current_time.year ↩
      +YEAR1);
  lcd_write_string(2, 4, line);
  lcd_draw_box(4,28,81,44, PIXEL_OFF);

  //highlight day
  lcd_draw_box(11,31,23,40, PIXEL_XOR);

/* //debug
  sprintf(line, "day p=%d m=%d", mode_prev, mode);
  lcd_write_string(0,1, line);
  usart1_send_string(line);
*/
  lcd_update();
}


/*****************************************************************************
* Function Name : inc_day
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void inc_day(void){
  unsigned char line[14] = "";

  if(++current_time.day == 32){ //count days up to 1 month
    current_time.day = 1; //reset days
  }
  else if (current_time.day == 31){ //for 30 day months
    if ((current_time.month == 4) || (current_time.month == 6) ||
    (current_time.month == 9) || (current_time.month == 11)){
      current_time.day=1;
    }
  }
  else if (current_time.day == 30){ //for month = Feb and leap year
    if(current_time.month==2){
      current_time.day=1;
    }
  }
  else if ((current_time.day==29) && !(LEAP(current_time.year+YEAR1))){
    if(current_time.month==2){
     current_time. day=1;
    }
```

```
  }


  lcd_clear_line(4);
  lcd_clear_box(11,31,23,40); //remove highlight

  sprintf(line, "%02d-%3s-%04d", current_time.day, month_name[current_time.month-1],current_time.year ↩
      +YEAR1);
  lcd_write_string(2, 4, line);

  //highlight day
  lcd_draw_box(11,31,23,40, PIXEL_XOR);

  lcd_update();
}


/*****************************************************************************
* Function Name : dec_day
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void dec_day(void){
  unsigned char line[14] = "";

 if(--current_time.day == 0){current_time.day = 1;}

  lcd_clear_line(4);
  lcd_clear_box(11,31,23,40); //remove highlight
  sprintf(line, "%02d-%3s-%04d", current_time.day, month_name[current_time.month-1],current_time.year ↩
      +YEAR1);
  lcd_write_string(2, 4, line);

  //highlight day
  lcd_draw_box(11,31,23,40, PIXEL_XOR);

  lcd_update();
}


//--------------------//
//year settings
//--------------------//
/*****************************************************************************
* Function Name : set_year
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void set_year(void){
  unsigned char line[14] = "";
```

```
  lcd_clear_line(2);
  lcd_write_string(0,2, "Set year");
  lcd_clear_box(5,29,80,43);

  sprintf(line, "%02d-%3s-%04d", current_time.day, month_name[current_time.month-1],current_time.year
      +YEAR1);
  lcd_write_string(2, 4, line);

  //highlight year
  lcd_draw_box(53,31,77,40, PIXEL_XOR);

  lcd_update();
}


/*****************************************************************************
* Function Name : inc_year
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void inc_year(void){
  unsigned char line[14] = "";

  if(++current_time.year == 255){current_time.year = 0;}

  lcd_clear_line(4);
  lcd_clear_box(53,31,77,40); //remove highlight

  sprintf(line, "%02d:%02d", current_time.hour,current_time.min);
  lcd_write_string(3, 3, line);
  sprintf(line, "%2d-%3s-%04d", current_time.day, month_name[current_time.month-1],current_time.year
      +YEAR1);
  lcd_write_string(2, 4, line);

  //highlight year
  lcd_draw_box(53,31,77,40, PIXEL_XOR);

  lcd_update();
}


/*****************************************************************************
* Function Name : dec_year
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void dec_year(void){
  unsigned char line[14] = "";

  if(--current_time.year == 0 ){current_time.year = 1;}
```

```
  lcd_clear_line(4);
  lcd_clear_box(53,31,77,40); //remove highlight

  sprintf(line, "%2d-%3s-%04d", current_time.day, month_name[current_time.month-1],current_time.year ↩
      +YEAR1);
  lcd_write_string(2, 4, line);

  //highlight year
  lcd_draw_box(53,31,77,40, PIXEL_XOR);

  lcd_update();
}



/*****************************************************************************
* Function Name : log_summary_display
* returns : none
* arguments : none
* Description : display summary of logged data
* Notes :
*****************************************************************************/
void log_summary_display(void){
  unsigned int max_value = 0;
  unsigned int last_value = 0;
  unsigned char line[14] = "";
  // unsigned char debug[40] = "";
  static unsigned int* current; //= data_pointer-1;

  current = file_current;
  file_prev = current;

  //sprintf(debug, "\rpd-%p %d, pv-%p %d",data_pointer, *data_pointer,
  // file_current,*file_current);
  //usart1_send_string(debug);

    last_value = *file_current;

    //moving backward, find max value until we hit mark2
    while(*file_current != MARK2){
      if(*file_current > max_value) {max_value = *file_current;}
      file_current--;
    }

  //display file id info
   sprintf(line, "%4d of %d", *(file_current-10),num_files);
   lcd_write_small_string(5,5,line);

   //display max and last values
   sprintf(line, "Last:%4d nA", last_value);
   lcd_write_string(0,3,line);

   sprintf(line, "Max:%4d nA", max_value);
   lcd_write_string(0,2,line);
```

```
    //get duration, interval and potential info for display
    sprintf(line, "%4dmV %2ds %2dm", *(file_current-9),*(file_current-8),
                                     *(file_current-7));

  lcd_write_string(0,1,line);


  //get time and date info for display
  sprintf(line, "%02d:%02d, %3s %02d", *(file_current-3),*(file_current-2),
                                       month_name[*(file_current-5)-1],
                                       *(file_current-4));
  lcd_write_string(0,0,line);

  //update global viewing variables
  if(*(file_current-10) == 1){file_prev = data_pointer-1;}
  file_current -= 11; //stop at begin file marker


  lcd_update();

  //sprintf(debug, "\rfp-%p %d, fc-%p %d",file_prev, *file_prev,
  // file_current,*file_current);
  //usart1_send_string(debug);
}




/*****************************************************************************
* Function Name : log_view
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void log_view(void){

  lcd_clear();
  if(num_files == 0){
    lcd_write_string(1, 2, "No files to");
    lcd_write_string(4, 3, "view");
    usart1_send_string("No files to view");
    }
  else{
    //set file_current to the eof (current data_pointer)
    file_current = data_pointer - 1;
    log_summary_display();
    view_file_num = 1;
    //file_prev = data_pointer - 1;
  }
}


/*****************************************************************************
```

```
* Function Name : log_view_next
* returns : none
* arguments : none
* Description :
* Notes :
*******************************************************************************/
void log_view_next(void){
  //unsigned char line[14] = "";

  lcd_clear();

  if(num_files == 0){
    lcd_write_string(1, 2, "No files to");
    lcd_write_string(4, 3, "view");
    lcd_update();
    }
  else{
    if(view_file_num == num_files){
      return;
    }
    else{
      file_current--;
      log_summary_display();
      view_file_num++;
    }
  }
}


/*******************************************************************************
* Function Name : log_view_prev
* returns : none
* arguments : none
* Description :
* Notes :
*******************************************************************************/
void log_view_prev(void){
  //unsigned char line[14] = "";

  lcd_clear();

  if(num_files == 0){
    lcd_write_string(1, 2, "No files to");
    lcd_write_string(4, 3, "view");
    }
  else{
    if(view_file_num == 1){
      file_current = data_pointer - 1;
      log_summary_display();
      view_file_num = 1;
    }
    else{
      file_current = file_prev;
      log_summary_display();
      view_file_num--;
```

```
    }
  }
  lcd_update();

}


/*****************************************************************************
* Function Name : log_delete
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void log_delete(void){
  //lcd_clear_line(2);
  lcd_clear();

  if(num_files == 0){
    lcd_write_string(2, 2, "No files to");
    lcd_write_string(4, 3, "delete");
    }
  else{
    lcd_write_string(2, 2, "Delete All");
    lcd_write_string(4, 3, "Files?");
  }
  lcd_update();
}


/*****************************************************************************
* Function Name : log_delete_confirm
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void log_delete_confirm(void){
  lcd_clear();

  if(num_files == 0){
    lcd_write_string(2, 2, "No files to");
    lcd_write_string(4, 3, "delete");
  }
  else{
    lcd_write_string(1, 3, "deleting...");
    lcd_update();

    data_log_erase();
    num_files = 0;
    mem_left = (unsigned int)DATA_MEM;
    data_pointer = (unsigned int *)DATA_START;
    view_file_num = 0;
    file_current = file_prev = data_pointer;
    delay_ms(1000);
```

```
    lcd_write_string(2, 2, "All files");
    lcd_write_string(3, 3, "deleted");
  }
  lcd_update();
  delay_ms(3000);
  log_select_delete();
}


/*****************************************************************************
* Function Name : log_delete_abort
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void log_delete_abort(void){
  //lcd_clear_line(2);
  lcd_clear();

   if(num_files == 0){
    lcd_write_string(2, 2, "No files to");
    lcd_write_string(4, 3, "delete");
    }
  else{
    lcd_write_string(1, 2, "Delete action");
    lcd_write_string(3, 3, "cancelled");
  }
  lcd_update();
  delay_ms(3000);
  log_select_delete();
}


/*****************************************************************************
* Function Name : log_select_view
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void log_select_view(void){
  unsigned char line[14]="";

  lcd_clear();
  lcd_write_string(2, 0, "Data Logs");
  lcd_draw_box(0,0,83,8, PIXEL_XOR);

  if(num_files == 0){
    lcd_write_string(1, 2, "No data files");
    lcd_write_string(3, 3, "present");
    }

  else{
```

```
    if(num_files == 1){lcd_write_string(1, 2, "1 Data File");}
    else{
      sprintf(line, "%d Files ", num_files);
      lcd_write_string(1, 2, line);
    }
    lcd_write_string(2, 3, "> View");
    lcd_write_string(4, 4, "Delete");
  }
  lcd_update();
}


/*****************************************************************************
* Function Name : log_select_delete
* returns : none
* arguments : none
* Description :
* Notes :
*****************************************************************************/
void log_select_delete(void){
  unsigned char line[14]="";

  lcd_clear();
  lcd_write_string(2, 0, "Data Logs");
  lcd_draw_box(0,0,83,8, PIXEL_XOR);

  if(num_files == 0){
    lcd_write_string(1, 2, "No data files");
    lcd_write_string(3, 3, "present");
    }

  else{
    if(num_files == 1){lcd_write_string(1, 2, "1 Data File");}
    else{
      sprintf(line, "%d Files ", num_files);
      lcd_write_string(1, 2, line);
    }
    lcd_write_string(4, 3, "View");
    lcd_write_string(2, 4, "> Delete");
  }
  lcd_update();
}


#endif //miniEC_fsm_C
```

A.2.4 Interface to global definitions

Listing A.6: Interface global definitions

```
#ifndef miniEC_globals_H
#define miniEC_globals_H

/*****************************************************************************
 *
 * $Workfile: globals.h $
 *
 * Definitions and interface to functions available to modules.
 *
 *
 *
 *
 * No guarantees, warrantees, or promises, implied.
 *
 *****************************************************************************/
#include <io.h>
#include <iomacros.h>
#include <msp430/iostructures.h>
#include <sys/inttypes.h>
#include "rtc.h"

#define ON 1
#define OFF 0

#define BUT1 P1IN & BIT4
#define BUT2 P1IN & BIT5
#define BUT3 P1IN & BIT6
#define BUT4 P1IN & BIT7
#define LED port1.out.pin0

#define B1 (1<<4)
#define B2 (1<<5)
#define B3 (1<<6)
#define B4 (1<<7)

#define LED port1.out.pin0

#define VERSION "5C 1.01"
#define VERSION_DATE __DATE__
#define VERSION_TIME __TIME__
#define PRODUCT_ID 0x0403
#define VENDOR_ID 0x6001
#define SERIAL_ID "ECPT5H6J"
#define CHANNELS 5

#define MARK1 11111
#define MARK2 21111
#define MARK3 31111

#define INFO_A 0x1080 //start of segment A in information memory
#define INFO_B 0x1000 //start of segment B in information memory
```

```
#define DATA_START 0x7D00 //start of data memory range
#define DATA_END 0xFDFF //end of data memory range
#define DATA_MEM DATA_END - DATA_START //total data memory


//-----------------------------------//
//data structures
//-----------------------------------//
//parameters for a test protocol
typedef struct{
  unsigned int Interval; //interval in seconds
  unsigned int Duration; // duration in minutes
  unsigned int Potential; //potential in mV
  //unsigned int Repeat; //run every 'Repeat' hours
  //unsigned char Range; //1=0-1000nA, 2=0-1000uA
  unsigned char ID; //1 to 5
  unsigned char name[10]; //name of protocol
}test_entry;


//prototype functions
void __inline__ delay(register unsigned int n);
void delay_ms(unsigned int value);
critical void fll_adjust(unsigned short multiplier);


/* ======================================== */
// Variables
/* ======================================== */
//timing
extern unsigned int interval;
extern unsigned long ntimes;
extern unsigned long nticks;
extern unsigned int duration;
extern unsigned long dtimes;
extern unsigned long dticks;
extern unsigned int dsec;
extern unsigned int dmin;
extern time_record test_start;
extern time_record current_time;
extern time_record alarm_time;
extern unsigned long repeat_time;

//measurement variables
extern unsigned int potential;
extern unsigned int gain;
extern unsigned char protocol;

extern unsigned int result[CHANNELS];
extern unsigned int result_max[CHANNELS];
extern unsigned long result_avg[CHANNELS];
extern unsigned long result_sum[CHANNELS];

// storage
extern unsigned char num_files;
```

```
extern unsigned int mem_left;
extern unsigned int *data_pointer;
extern unsigned short monitor_flag;

/*
//Alerts removed for now.

void alarm_set(void);
void alarm_off(void);
void set_repeat(void);
void stop_repeat(void);

void alert_memory(int memory);
void alert_battery(void);
void alert_pc_connected(void);
void alert_pc_disconnected(void);

*/

#endif //miniEC_globals_H
```

A.2.5 Global definitions

Listing A.7: Global definitions module

```
#ifndef miniEC_globals_C
#define miniEC_globals_C

#include <io.h>
#include "globals.h"

/*****************************************************************************
 Utility functions for all modules
 *****************************************************************************/


// simple delay
//=============
//counting clock cycles is not as accurate as using a timer.Current clock timing
//is only accurate to seconds. This simple delay is good enough for short delays.
//delay = 3 CPU cycles per count plus a few cyles for function call and return.
//delay ~= 6+3*n clock cycles
//delay(n) ~= 6+3*n counts


void __inline__ delay(register unsigned int n){
  __asm__ __volatile__ (
              "1: \n"
              " dec %[n] \n"
              " jne 1b \n"
              : [n] "+r"(n)
              );
}




/*****************************************************************************
 * Function Name: delay_ms

 delay in miliseconds is dependent on CPU clock. Adjust as needed
 for mclk=800k, 1ms ~= 264 counts
 for mclk=1MHz, 1ms ~= 331 counts
 *****************************************************************************/
void delay_ms(unsigned int value){
  volatile int j;

  for(j=value; j>0; j--){delay(2665);}
}

#endif //miniEC_globals_C
```

A.2.6 Interface to hardware module

Listing A.8: Interface to hardware module

```
#ifndef miniEC_hardware_H
#define miniEC_hardware_H

/*****************************************************************************
 *
 * $Workfile: hardware.h $
 *
 * This module provides an interface to routines which access hardware peripherals
 * internal to the MSP430 microcontroller. These include ADC, DAC, flash memory,
 * general I/O and USART.
 *
 *
 * No guarantees, warrantees, or promises, implied.
 *
 *****************************************************************************/
#include <signal.h>
#include "globals.h"

//definitions
#ifndef ON
#define ON 1
#endif
#ifndef OFF
#define OFF 0
#endif

#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif

#define F0 16
#define F1 8

//macros
#define RX_CMD(a,b) (rxBuffer[0]==a && rxBuffer[1]==b)
#define FLL_MULTIPLIER(smclk, aclk) ((smclk)/(aclk))

critical void fll_adjust(unsigned short multiplier);
void beep(unsigned short multiplier);

//Flash memory
void flash_erase_segment(void *address);
void flash_write(void *dst, const void *src, unsigned int size);
void flash_write_word(void *dst, const void *src, unsigned int size);

void flash_clear_memory(void);
void write_SegA(char value);
```

```
//analog and digital convertors
void DAC_init(void);
void DAC_stop(void);
void DAC_start(int mv);
void ADC_init(void);
void ADC_read(void);
void ADC_stop(void);
void end_measure(void);
interrupt(ADC_VECTOR) adc12_isr (void);

//usart1
void usart1_init(void);
void usart1_send_char(char c);
void usart1_send_string(char msg[]);
void usart1_send_int(int value);
void usart1_rx_isr_enable(void);
void usart1_rx_isr_disable(void);

//interrupt(UART0RX_VECTOR) usart1_rx_isr(void);


//timer
void start_interval_timer(int ival, int dur);
void stop_interval_timer(void);
interrupt(WDT_VECTOR) watchdog_timer_isr(void);

//ports
void ports_init(void);

#endif //miniEC_harware_H
```

A.2.7 Hardware module

Listing A.9: Hardware module

```
#ifndef miniEC_hardware_C
#define miniEC_hardware_C

/*****************************************************************************
 *
 * $Workfile: hardware.c $
 *
 * This module encapsulates routines to access hardware peripherals internal
 * to the MSP430 microcontroller. These include ADC, DAC, flash memory,
 * general I/O and USART.
 *
 *
 * No guarantees, warrantees, or promises, implied.
 *
 *****************************************************************************/

#include <io.h>
#include <iomacros.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "hardware.h"
#include "fsm.h"




/*****************************************************************************
* Notes  : from mspgcc library of useful routines
*****************************************************************************/
// set DCO to selected frequency
// MCLK = multiplier*ACLK

critical void fll_adjust(unsigned short multiplier) {
  unsigned short compare;
  unsigned short old_capture = 0;

  CCTL2 = CCIS0|CM0|CAP; // Define CCR2, CAP, ACLK
  TACTL = TASSEL_SMCLK|TACLR|MC1; // SMCLK, continous mode
  while (1) {
    while (!(CCTL2 & CCIFG)) {} // Wait until capture occured!
    CCTL2 &= ~CCIFG; // Capture occured, clear flag
    compare = CCR2 - old_capture; // SMCLK difference
    old_capture = CCR2; // Save current captured SMCLK
    if (multiplier == compare) {
      break; // if equal, leave "while(1)"
    } else if (multiplier < compare) { // DCO is too fast, slow it down
      DCOCTL--;
      if (DCOCTL == 0xff) { // Did DCO role under?
    BCSCTL1--; // Select next lower RSEL
```

```
      }
    } else {
      DCOCTL++;
      if (DCOCTL == 0x00) { // Did DCO role over?
    BCSCTL1++; // Select next higher RSEL
      }
    }
  }
  CCTL2 = 0; // Stop CCR2 function
  TACTL = 0; // Stop Timer_A
}

/*-------------------------------------------------------------------------
Port setup. Settings may be modified by particular modules.
-------------------------------------------------------------------------*/

void ports_init(void){
  //port1 setup:
  //P1.0 LED, P1.1,2 buzzer outputs
  //P1.3 rs232 input, p1.4 - 1.7 push button inputs
  P1DIR |= 0x07;
  LED = OFF;
  P1IFG = 0x00; // clear pending interrupts
  P1IES = B1 + B2 + B3 + B4; // interrupt on falling edge
  P1IE = B1 + B2 + B3 + B4; // enable push button interrupts

  //port2 setup:P
  //p2.1, low battery input, rest are unused outputs
  P2SEL |= 0x00;
  P2DIR |= 0xFE;

  //port3 setup
  //lcd spi and usart1
  P3SEL |= 0xCA;
  P3DIR |= 0x1;

  //port4 setup:
  //P4.1,LCD spi select
  //P4.0,P4.2, dataflash spi select and reset
  P4SEL |= 0x00;
  P4DIR |= 0xFF;

 //port5: not used, set all to outputs
  //P5SEL = 0xFF;
  P5DIR = 0xFF;

  //port6 setup:
  //P6.1 - P6.5, ADC, sensor inputs
  //P6.6, DAC, potential output
  P6SEL |= BIT1 + BIT2 + BIT3 + BIT4 + BIT5;
  P6DIR |= BIT0 + BIT6 + BIT7;
  P6OUT = 0x00;
}
```

```
/*------------------------------
timerA and buzzer
 ------------------------------*/
/*
void beep(unsigned short multiplier){
  if(multiplier < 1){multiplier = 1;}
  if(multiplier > 3){multiplier = 3;}

  TACCR0 = 0;
  TACCR1 = 0;
  TACCTL0 = OUTMOD_4;
  TACCTL1 = OUTMOD_4;
  TACCR0 = F0*multiplier;
  TACCR1 = F1*multiplier;

  delay(0xffff);
  delay(0xffff);

  TACCR0 = OFF;
  TACCR1 = OFF;
}
*/

/*------------------------------
 Flash memory
 ------------------------------*/
/*****************************************************************************
* Function name : flash_write
* returns :
* arg1 :
* arg2 :
* Description :
* Notes  : FCTL2 must be initialized before calling this function.
* Might want to disable interrupts before calling function and
* enable after writing.
*****************************************************************************/
void flash_write(void *dst, const void *src, unsigned int size) {
//void flash_write(void *dst, const void *src, unsigned int size) {
    FCTL3 = FWKEY; // unlock
    FCTL1 = FWKEY|WRT; // select write

    while (size) {
        size--;
        ((unsigned char *)dst)[size] = ((unsigned char *)src)[size]; // copy
        while (FCTL3 & BUSY) {} // wait until write is completed
    }
    FCTL1 = FWKEY; // disable flash writing
    FCTL3 = FWKEY|LOCK; // lock
    return;
}

/*****************************************************************************
* Function name : flash_write_word
* returns : none :
```

```
* arg1 : *dst, location in flash to write to
* arg2 : *src, pointer to value to be written
* arg3 : size, length of the of the value to be written
* Description :
* Notes  : FCTL2 must be initialized before calling this function. The
* data must be word aligned. Might also want to disable
* interrupts before calling function and enable after writing.
*******************************************************************************/

void flash_write_word(void *dst, const void *src, unsigned int size) {
    FCTL3 = FWKEY; // unlock flash
    FCTL1 = FWKEY|WRT; // select write
    //calculate size in words: divide by two (round down size if it is odd)
    size >>= 1;

    while (size) {
        size--;
        ((unsigned short *)dst)[size] = ((unsigned short *)src)[size]; // copy
        while (FCTL3 & BUSY) {} // wait until write is completed
    }
    FCTL1 = FWKEY; // disable flash writing
    FCTL3 = FWKEY|LOCK; // lock flash
    return;
}


/*******************************************************************************
* Function name :
* returns :
* arg1 :
* arg2 :
* Description : General segment erase function.
* Notes  : FCTL2 must be initialized before calling this function.
* For F2xx devices, the LOCKA bit must be cleared as well
* to erase segment A. User must be careful not to
* erase program memory!
*******************************************************************************/
void flash_erase_segment(void *address) {
    FCTL3 = FWKEY; // unlock
    FCTL1 = FWKEY|ERASE; // select segment erase
    *(unsigned short*)address = 0xffff; // erase
    FCTL1 = FWKEY; // disable flash writing
    FCTL3 = FWKEY|LOCK; // lock
    return;
}


/*******************************************************************************
* Function name :
* returns :
* arg1 :
* arg2 :
* Description :
* Notes  : FCTL2 must be initialized before calling this function.
*******************************************************************************/
```

```
void flash_clear_memory(void){
  unsigned int *address = (unsigned int *)DATA_START;

  dint();
  do
    {
      if ((unsigned int)address & 0x1000)
    {
      LED |= ON;

    }
      else
    LED = OFF;

      FCTL1 = FWKEY + ERASE;
      *address = 0x00;

      address += 0x0100;
    } while (address < (unsigned int *)DATA_END);
  FCTL3 = FWKEY | LOCK; // lock the flash again
  eint(); //enable interrupts.
  return;
}

/*****************************************************************************
* Function name :
* returns :
* arg1 :
* arg2 :
* Description :
* Notes  : FCTL2 must be initialized before calling this function.
*****************************************************************************/
void write_SegA(char value)
{
  char *Flash_ptr; // Flash pointer
  unsigned int i;

  Flash_ptr = (char *) 0x1080; // Initialize Flash pointer
  FCTL1 = FWKEY + ERASE; // Set Erase bit
  FCTL3 = FWKEY; // Clear Lock bit
  *Flash_ptr = 0; // Dummy write to erase Flash segment

  FCTL1 = FWKEY + WRT; // Set WRT bit for write operation

  for (i=0; i<128; i++)
  {
    *Flash_ptr++ = value; // Write value to flash
  }

  FCTL1 = FWKEY; // Clear WRT bit
  FCTL3 = FWKEY + LOCK; // Reset LOCK bit
  return;
}
```

```
/*------------------------------
 usart1
 ------------------------------*/
/*****************************************************************************
* Function name : usart1_init
* returns :
* arg1 :
* arg2 :
* Description :
* Notes  : 2400 or 9600 baud rate
*****************************************************************************/
void usart1_init(void){
  P3SEL |= 0xC0; // P3.6,7 = USART1 TXD/RXD
  UTCTL1 |= SSEL0; // UCLK = ACLK
  UCTL1 |= CHAR; // 8-bit character *SWRST*

  //UBR01 = 0x0D; // 32k/2400 - 13.65
  //UMCTL1 = 0x6B; // Modulation for 2400
  UBR01 = 0x03; // 32k/9600 - 3.41
  UMCTL1 = 0x4A; // Modulation for 9600

  UBR11 = 0x00; // Modulation
  UCTL1 &= ~SWRST; // Initialize USART state machine
  ME2 |= UTXE1 + URXE1; // Enable USART1 TXD/RXD
  IE2 |= URXIE1; // Enable USART1 RX interrupt
}



/*---------------------------------
-//Transmit 1 byte
---------------------------------*/
void usart1_send_char(char c){
  while (!(IFG2 & UTXIFG1)); // USART1 TX buffer ready
  TXBUF1 = c;
}

/*---------------------------------
-//Transmit messages through usart1
---------------------------------*/
void usart1_send_string(char msg[]){
  while (*msg != 0){
     while ((IFG2 & UTXIFG1) == 0); //USART TX buffer ready?
     TXBUF1 = *msg; //send character to TXBUF
     msg++; //increment pointer
  }
}

/*---------------------------------
-//send numbers
---------------------------------*/
void usart1_send_int(int value){
  char buffer[6];

  if(value == 0){
```

```
    usart1_send_char('0');
    usart1_send_char(' ');
  }
  else {
    sprintf(buffer, "%d ", value);
    usart1_send_string(buffer);
  }
}


/*-------------------------------
-Enable usart1 interrupt
-------------------------------*/
void usart1_rx_isr_enable(void)
{
  IE2 |= URXIE1;
}

/*-------------------------------
--Disable usart1 interrupt
-------------------------------*/
void usart1_rx__isr_disable(void)
{
  IE2 &= ~URXIE1;
}

/*-----------------------------
 analog and digital conversions
 -----------------------------*/
/*****************************************************************************
* Description : //DAC, initialize and calibrate  :

void DAC_init(void){
  ADC12CTL0 |= REF2_5V + REFON; //2.5V reference on
  DAC12_0CTL = DAC12IR + DAC12AMP_5 + DAC12ENC;//p6.6, gain=1
  DAC12_0CTL |= DAC12CALON; //calibrate dac output
  delay(0xffff); //wait for calibration
  delay(0xffff); //wait for calibration
}

/*****************************************************************************
// Switch off DAC12
*****************************************************************************/
void DAC_stop(void){
  //DAC12_0CTL = 0;
  //DAC12_0DAT = 0;

  ADC12CTL0 = 0; // switch off ADC12 ref voltage
  DAC12_0CTL &= ~DAC12ENC; // disable DAC12 conversion
  DAC12_0CTL = 0; // switch off DAC12
}

/*****************************************************************************
// Write a value for DAC to output
// 0xffff = 2500mV
```

```
// unsigned int v = 0x28F;
// => dac value = mv*0xffff/2500
// 0xffff/2500 = 26.2, use 26
// scale to fit 0 to 4095 instead of 0 to 65335
// avoids floating point math and gives accurate result
*****************************************************************************/
void DAC_start(int mv){


  unsigned long v = mv*26/16;

  DAC12_0DAT = v;
}


/*****************************************************************************
 Initialize ADC
 *****************************************************************************/
void ADC_init(void){
  ADC12CTL0 = ADC12ON+MSC+REFON+REF2_5V+SHT0_8; // Setup ADC (sht=16 adcclk cycles)
  ADC12CTL1 = SHP + CONSEQ_3; // Use sampling timer, repeated sequence
  ADC12MCTL0 = INCH_1+SREF_1; // start with channel A1
  ADC12MCTL1 = INCH_2+SREF_1; // internal ref+, channel = A2
  ADC12MCTL2 = INCH_3+SREF_1; // internal ref+, channel = A3
  ADC12MCTL3 = INCH_4+SREF_1; // internal ref+, channel = A4
  ADC12MCTL4 = INCH_5+SREF_1+EOS;; // internal ref+, channel = A5 end seq.
  ADC12IE = BIT4; // Enable ADC12IFG.4
  ADC12CTL0 |= ENC; // Enable conversisons
  delay(0xFFFF); // delay
}

/*****************************************************************************
-//ADC, stop
*****************************************************************************/
void ADC_stop(void){
  ADC12CTL0 &= ~ENC; // Disable ADC
  ADC12CTL1 &= ~CONSEQ_3; // Stop conversions
  while(ADC12CTL0 & ADC12BUSY); // wait for conversion to stop
  ADC12CTL0 &= ADC12CTL0; // Switch off ADC & ref voltage
  ADC12IFG = 0; //clear any pending interrupts
}


/*****************************************************************************
-ADC interrupt
*****************************************************************************/
interrupt(ADC_VECTOR) adc12_isr (void){
  volatile int numToAvg = 64;
  volatile int i;
  static unsigned long ADCsum[CHANNELS] = {0};


  //accumulate 64 readings for each channel
  while (numToAvg > 0){
```

```
    ADCsum[0] += ADC12MEM0; // Move results, IFG is cleared
    ADCsum[1] += ADC12MEM1; // Move results, IFG is cleared
    ADCsum[2] += ADC12MEM2; // Move results, IFG is cleared
    ADCsum[3] += ADC12MEM3; // Move results, IFG is cleared
    ADCsum[4] += ADC12MEM4; // Move results, IFG is cleared

    numToAvg--;
  }


  if (numtoAvg == 0){
    ADC12CTL0 &= ~ENC; // Stop conversions

    for(i=0;i<CHANNELS;i++){
      result_avg[i] = ADCsum[i] >> 6; //divide by 64(>>6)
      ADCsum[i] = 0; //reset
    }

    //reset
    numToAvg = 64;
  }

  LPM0_EXIT;
}


/*
interrupt(ADC_VECTOR) adc12_isr (void){

  result_avg[0] = ADC12MEM0; // Move results, IFG is cleared
  result_avg[1] = ADC12MEM1; // Move results, IFG is cleared
  result_avg[2] = ADC12MEM2; // Move results, IFG is cleared
  result_avg[3] = ADC12MEM3; // Move results, IFG is cleared
  result_avg[4] = ADC12MEM4; // Move results, IFG is cleared

  ADC12CTL0 &= ~ENC; // Stop conversion

  LPM0_EXIT;
}
*/

/*****************************************************************************
//ADC read
//For adc measurements using oversampling and averaging methods for noise
//suppression. ADC value then stored/displayed/tramsmited via usart
*****************************************************************************/
void ADC_read(void){
  volatile int channel = 0;


  ADC12CTL0 |= ENC; // Enable conversions
  ADC12CTL0 |= ADC12SC; // Start ADC sampling and conversion
  LPM0; // wait in LPM0 for conversion to finis
```

```
 // multiply by voltage reference, scale by 12-bits and
 // divide by gain (2M) to convert to current in nA
 // value = ( unsigned int )(((result*2500)>>12)/2);

  for(channel=0; channel<CHANNELS; channel++){
    result_avg[channel] *= 1250; //multiply by (voltage reference/gain)
    //result_avg[channel] = result_avg[channel]/gain; //divide by gain
    result_avg[channel] = result_avg[channel] >> 12; //scale by 12 bits
    result[channel] = (unsigned int)result_avg[channel];
    result_avg[channel] = 0; //added in V2
  }

}


/*****************************************************************************
 * Function Name: stop measureing
--defunct function. Not used anymore
 *****************************************************************************/
void end_measure(void){
  stop_interval_timer();
  ADC_stop();
  DAC_stop();
}


/*****************************************************************************
-Using the watchdog timer for interval timing of samples
-Sample Intervals = 1 to 60 sec, duration in minutes
=watchdog is clocked by ACLK
 *****************************************************************************/
void start_interval_timer(int ival, int dur){
  nticks = 0;
  dticks = 0;
  dsec = dmin = 0;

  ntimes = ival;
  dtimes = dur*60;

  WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer
  IE1 &= ~WDTIE; // disable WDT interrupt
  WDTCTL = WDT_ADLY_1000; // WDT 1s, ACLK, interval mode
  IE1 |= WDTIE; // Enable WDT interrupt
}


void stop_interval_timer(void){
  WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer
  IE1 &= ~WDTIE; // disable WDT interrupt
}
```

```
/*****************************************************************************
- interval timer interrupt
- when nticks == ntimes, put FIVAL in the event tray
- when duration dtimes is reached, put FDONE in the event tray
 *****************************************************************************/
interrupt(WDT_VECTOR) interval_timer_isr(void){
  if(++nticks == ntimes){
     event_tray |= FIVAL;
    nticks = 0;
  }

  if(++dticks > dtimes){
    event_tray |= FDONE;
    dticks = 0;
  }

  if (++dsec == 60){
    dsec = 0;
    dmin++;
  }

  LPM3_EXIT;
}


#endif //miniEC_harware_C
```

A.2.8 Interface to icon definitions

Listing A.10: Interface to icon definitions

```
#ifndef miniEC_icons_H
#define miniEC_icons_H

/*****************************************************************************
 *
 * $Workfile: icons.c $
 *
 * This module contains the structures and function prototypes for
 * drawing the miniEC's icons.
 *
 *
 * No guarantees, warrantees, or promises, implied.
 *
 *****************************************************************************/

//prototype functions
void display8x8(int x, int y, const unsigned char* bitmap);
void display16x16(int x, int y, const unsigned char* bitmap);
void display24x24(int x, int y, const unsigned char* bitmap);

//down arrow
extern const unsigned char down[];
extern const unsigned char down1[];

//up arrow
extern const unsigned char up[];
extern const unsigned char up1[];

//clock bitmap
extern const unsigned char clock24[];

//doc bitmap
extern const unsigned char file24[];

//setup bitmap 24x24, 74 bytes
extern const unsigned char setup24[];

//idle bitmap 24x24, 74 bytes
extern const unsigned char idle24[];

//clock bitmap
extern const unsigned char clock16[];

extern const unsigned char setup16[];
extern const unsigned char file16[];

#endif //miniEC_icons_H
```

A.2.9 Icon definitions

Listing A.11: Icon definitions

```
#ifndef miniEC_icons_C
#define miniEC_icons_C

/*****************************************************************************
 *
 * $Workfile: icons.c $
 *
 * This module provides routines to draw icons on the miniEC's LCD screen
 *
 *
 * No guarantees, warrantees, or promises, implied.
 *
 *****************************************************************************/
#include "pcd8544.h"
#include "icons.h"

//initialize bitmaps for various icons
const unsigned char down[] = {0x00,0x06,0x1E,0x7E,0x1E,0x06,0x00,0x00};
const unsigned char down1[] = {0x03,0x0F,0x3F,0xFF,0x3F,0x0F,0x03,0x00};

//up arrow
const unsigned char up[] = {0x00,0x60,0x78,0x7E,0x78,0x60,0x00,0x00};
const unsigned char up1[] = {0xC0,0xF0,0xFC,0xFF,0xFC,0xF0,0xC0,0x00};

//clock bitmap
const unsigned char clock24[] = {
0x00,0x00,0x00,0x80,0xC0,0x60,0xB0,0x18,
0x08,0x08,0x0C,0x0C,0x9C,0x0C,0x08,0x08,
0x18,0xB0,0xE0,0xC0,0x80,0x00,0x00,0x00,
0x00,0x00,0x3C,0xFF,0xCB,0x00,0x00,0x00,
0x00,0xC0,0x60,0x20,0x3F,0x00,0x00,0x00,
0x00,0x00,0x00,0xCB,0xFF,0x3C,0x00,0x00,
0x00,0x00,0x00,0x01,0x03,0x06,0x0D,0x18,
0x10,0x10,0x30,0x38,0x30,0x30,0x10,0x10,
0x18,0x0E,0x06,0x03,0x01,0x00,0x00,0x00
};

//doc bitmap
const unsigned char file24[] = {
0x00,0x00,0x00,0x00,0xF8,0xFC,0x0C,0x0C,
0x0C,0x0C,0x0C,0x0C,0x0C,0xFC,0xF8,0xF0,
0xE0,0xC0,0x80,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0x00,
0x90,0x90,0x90,0x90,0x90,0x93,0x93,0x93,
0x03,0x03,0x03,0xFF,0xFE,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x3F,0x3F,0x30,0x30,
0x34,0x34,0x34,0x34,0x34,0x34,0x34,0x34,
0x30,0x30,0x30,0x3F,0x1F,0x00,0x00,0x00
};

//setup bitmap 24x24, 74 bytes
```

```
const unsigned char setup24[] = {
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,
0xF0,0xF0,0xE0,0x00,0x00,0x00,0x00,0x00,
0x00,0xC0,0xC0,0xE0,0x60,0x00,0x00,0x00,
0x00,0x00,0x02,0x07,0x0E,0x0E,0x0E,0x8E,
0xCF,0xE7,0xF7,0xFC,0xF0,0x58,0xEC,0xC6,
0x03,0x01,0x01,0x01,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x1C,0x1E,0x3F,0x1F,
0x0F,0x07,0x03,0x01,0x00,0x00,0x02,0x07,
0x0F,0x1E,0x0C,0x00,0x00,0x00,0x00,0x00
};

//idle bitmap 24x24, 74 bytes
const unsigned char idle24[] = {
0x02,0x07,0x1E,0x18,0x30,0x38,0x40,0x40,
0x00,0x02,0xCF,0xCF,0xCF,0xC6,0xC0,0xC0,
0x80,0x00,0x30,0x38,0x18,0x06,0x07,0x01,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,
0x80,0x9F,0x3F,0x3F,0x7F,0x7F,0x7F,0x7F,
0x3F,0x9C,0x00,0x00,0x00,0x00,0x00,0x00,
0xC0,0xC0,0xF0,0x78,0x3C,0x1E,0x06,0x07,
0x03,0x01,0x00,0x00,0x00,0x00,0x00,0x00,
0x01,0x03,0x07,0x0E,0x1E,0x3C,0x78,0xF0
};

//clock bitmap
const unsigned char clock16[] = {
0xC0,0xF8,0x7C,0xFE,0xE6,0xDE,0x9B,0x83,
0x8F,0x03,0x1B,0x0E,0x66,0x3C,0x38,0xF0,
0x07,0x1F,0x39,0x6D,0xE4,0xF0,0x91,0x81,
0xE1,0x83,0xF7,0xF6,0xEC,0x7D,0x3D,0x1F
};

const unsigned char setup16[] = {
0xFC,0x04,0x04,0x04,0x04,0x04,0x04,0xC2,
0xF2,0xE6,0x44,0x46,0xF9,0xCD,0xC7,0x00,
0xF1,0x99,0xCB,0x2A,0x2C,0xCF,0x9D,0xFB,
0x0E,0x15,0x1D,0x1B,0x0E,0x03,0x00,0x00
};

const unsigned char file16[] = {
0x00,0xFC,0x02,0xAA,0xAA,0xAA,0xAA,0xAA,
0xAA,0xAA,0x02,0xFC,0x00,0x00,0x00,0x00,
0x00,0x1F,0x20,0x2A,0x2A,0x2A,0x2A,0x2A,
0x2A,0x2A,0x20,0x1F,0x00,0x00,0x00,0x00
};

/*************************************************************************
* Function Name : display8x8
* returns : none
* arg1 : x coordinate
* arg2 : y coordinate
* arg3 : bitmap
* Description : Draws an 8 by 8 pixel bit map starting at the location
* indicated by x and y
```

```
* Notes :
*******************************************************************************/
void display8x8(int x, int y, const unsigned char* bitmap){
  unsigned int i;
  unsigned int idx = (x*48 + y*48*14)/8 ;

  for(i=0;i<8;i++){
    LcdMemory[idx] = bitmap[i];
    idx++;
  }
}

/*******************************************************************************
* Function Name : display16x16
* returns : none
* arg1 : x coordinate
* arg2 : y coordinate
* arg3 : bitmap
* Description : Draws a 16 by 16 pixel bit map starting at the location
* indicated by x and y
* Notes :
*******************************************************************************/
void display16x16(int x, int y, const unsigned char* bitmap){
  unsigned int i;
  unsigned int idx;


  //first row
  idx = (x*48 + y*48*14)/8 ;
  for(i=0;i<16;i++){
    LcdMemory[idx] = bitmap[i];
    idx++;
  }

  //second row
  idx = (x*48 + (1+y)*48*14)/8 ;
  for(i=16;i<32;i++){
    LcdMemory[idx] = bitmap[i];
    idx++;
  }
}

/*******************************************************************************
* Function Name : display24x24
* returns : none
* arg1 : x coordinate
* arg2 : y coordinate
* arg3 : bitmap
* Description : Draws a 24 by 24 pixel bit map starting at the location
* indicated by x and y
* Notes :
*******************************************************************************/
void display24x24(int x, int y, const unsigned char* bitmap){
  unsigned int i;
  unsigned int idx;
```

```
  //first row
  idx = (x*48 + y*48*14)/8 ;
  for(i=0;i<24;i++){
    LcdMemory[idx] = bitmap[i];
    idx++;
  }

  //second row
  idx = (x*48 + (1+y)*48*14)/8 ;
  for(i=24;i<48;i++){
    LcdMemory[idx] = bitmap[i];
    idx++;
  }

  //3rd row
  idx = (x*48 + (2+y)*48*14)/8 ;
  for(i=48;i<72;i++){
    LcdMemory[idx] = bitmap[i];
    idx++;
  }
}

#endif //miniEC_icons_C
```

A.2.10 Interface to LCD module

Listing A.12: Interface to LCD module

```
#ifndef miniEC_pcd8544_H
#define miniEC_pcd8544_H

/* ================================================
*
* pcd8544.c
*
* Driver for PCD8544 controlled LCDs
*
* Command codes
* ---------- function set
* 32 = basic instr set, chip is active, horizontial addressing
* 33 = extended instruction set
* 34 = vertical addressing
* 36 = power down mode
*
* ---------- basic instruction set
* 08 = display blank
* 09 = all display segments on
* 12 = normal mode
* 13 = inverse video mode
*
* ---------- extended instruction set
* set temperature coefficient, range 4 - 7
* set bias system, range 16 - 23
* set Vop 128-255
*
* ---------- LCD registers
*--set y address of ram 64+ 0<=y<=5
*--set x address of ram 128+ 0<=x<=83
*

* ================================================ */

#include "globals.h"

/* definitions */

#ifndef ON
#define ON 1
#endif
#ifndef OFF
#define OFF 0
#endif

#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif
```

```
#ifndef NULL
#define NULL 0
#endif

#define SEND_CMD 0
#define SEND_CHR 1

#define LCD_X_RES 84
#define LCD_Y_RES 48

#define PIXEL_OFF 0
#define PIXEL_ON 1
#define PIXEL_XOR 2

#define BOX_OFF 0
#define BOX_ON 1
#define BOX_XOR 2

#define FONT_0X 0
#define FONT_1X 1
#define FONT_2X 2

#define LCD_CACHE_SIZE ((LCD_X_RES * LCD_Y_RES) / 8)

//SPI
#define LCD_RES port4.out.pin1
#define LCD_CS port3.out.pin0

#ifndef SPI_MISO
#define SPI_MISO port3.out.pin1
#endif
#ifndef SPI_MOSI
#define SPI_MOSI port3.out.pin2
#endif
#ifndef SPI_SCKL
#define SPI_SCKL port3.out.pin3
#endif

//Variables
unsigned int LcdMemIdx; // LCD memory index
unsigned char LcdMemory[LCD_CACHE_SIZE]; // LCD memory

//Function prototypes
void lcd_init(void);
void lcd_clear(void);
void lcd_update(void);
void lcd_clear_line(unsigned char row);
void lcd_send(unsigned char data, unsigned char cd);

void lcd_write_char(unsigned char x, unsigned char y, unsigned char ch);
void lcd_write_string(unsigned char col, unsigned char row, unsigned char *dataPtr);
void lcd_write_small_char(unsigned char x, unsigned char y, unsigned char ch);
void lcd_write_small_string(unsigned char col, unsigned char row, unsigned char *dataPtr);
void lcd_write_2X_char(unsigned char x, unsigned char y, unsigned char ch);
```

```
void lcd_write_2X_string(unsigned char col, unsigned char row, unsigned char *dataPtr);

void lcd_set_pixel(unsigned char x, unsigned char y, unsigned char mode);
void lcd_set_xy(unsigned char x, unsigned char y );
void lcd_set_contrast(unsigned char contrast);
void lcd_fill_image(const unsigned char* bitmap);
void LcdLine(unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2, unsigned char mode );
void lcd_draw_box(unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2, unsigned char mode);
void lcd_clear_box(unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2);
void lcd_draw_line(unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2,unsigned char mode);

/*
* Standard ASCII characters in a 5x7 pixel font
* Characters 32-125 in ascii table
*/
static const unsigned char FontLookup [][5] =
{
    { 0x00, 0x00, 0x00, 0x00, 0x00 }, // sp
    { 0x00, 0x00, 0x2f, 0x00, 0x00 }, // !
    { 0x00, 0x07, 0x00, 0x07, 0x00 }, // "
    { 0x14, 0x7f, 0x14, 0x7f, 0x14 }, // #
    { 0x24, 0x2a, 0x7f, 0x2a, 0x12 }, // $
    { 0xc4, 0xc8, 0x10, 0x26, 0x46 }, // %
    { 0x36, 0x49, 0x55, 0x22, 0x50 }, // &
    { 0x00, 0x05, 0x03, 0x00, 0x00 }, // '
    { 0x00, 0x1c, 0x22, 0x41, 0x00 }, // (
    { 0x00, 0x41, 0x22, 0x1c, 0x00 }, // )
    { 0x14, 0x08, 0x3E, 0x08, 0x14 }, // *
    { 0x08, 0x08, 0x3E, 0x08, 0x08 }, // +
    { 0x00, 0x00, 0x50, 0x30, 0x00 }, // ,
    { 0x10, 0x10, 0x10, 0x10, 0x10 }, // -
    { 0x00, 0x60, 0x60, 0x00, 0x00 }, // .
    { 0x20, 0x10, 0x08, 0x04, 0x02 }, // /
    { 0x3E, 0x51, 0x49, 0x45, 0x3E }, // 0
    { 0x00, 0x42, 0x7F, 0x40, 0x00 }, // 1
    { 0x42, 0x61, 0x51, 0x49, 0x46 }, // 2
    { 0x21, 0x41, 0x45, 0x4B, 0x31 }, // 3
    { 0x18, 0x14, 0x12, 0x7F, 0x10 }, // 4
    { 0x27, 0x45, 0x45, 0x45, 0x39 }, // 5
    { 0x3C, 0x4A, 0x49, 0x49, 0x30 }, // 6
    { 0x01, 0x71, 0x09, 0x05, 0x03 }, // 7
    { 0x36, 0x49, 0x49, 0x49, 0x36 }, // 8
    { 0x06, 0x49, 0x49, 0x29, 0x1E }, // 9
    { 0x00, 0x36, 0x36, 0x00, 0x00 }, // :
    { 0x00, 0x56, 0x36, 0x00, 0x00 }, // ;
    { 0x08, 0x14, 0x22, 0x41, 0x00 }, // <
    { 0x14, 0x14, 0x14, 0x14, 0x14 }, // =
    { 0x00, 0x41, 0x22, 0x14, 0x08 }, // >
    { 0x02, 0x01, 0x51, 0x09, 0x06 }, // ?
    { 0x32, 0x49, 0x59, 0x51, 0x3E }, // @
    { 0x7E, 0x11, 0x11, 0x11, 0x7E }, // A
    { 0x7F, 0x49, 0x49, 0x49, 0x36 }, // B
```

```
{ 0x3E, 0x41, 0x41, 0x41, 0x22 }, // C
{ 0x7F, 0x41, 0x41, 0x22, 0x1C }, // D
{ 0x7F, 0x49, 0x49, 0x49, 0x41 }, // E
{ 0x7F, 0x09, 0x09, 0x09, 0x01 }, // F
{ 0x3E, 0x41, 0x49, 0x49, 0x7A }, // G
{ 0x7F, 0x08, 0x08, 0x08, 0x7F }, // H
{ 0x00, 0x41, 0x7F, 0x41, 0x00 }, // I
{ 0x20, 0x40, 0x41, 0x3F, 0x01 }, // J
{ 0x7F, 0x08, 0x14, 0x22, 0x41 }, // K
{ 0x7F, 0x40, 0x40, 0x40, 0x40 }, // L
{ 0x7F, 0x02, 0x0C, 0x02, 0x7F }, // M
{ 0x7F, 0x04, 0x08, 0x10, 0x7F }, // N
{ 0x3E, 0x41, 0x41, 0x41, 0x3E }, // O
{ 0x7F, 0x09, 0x09, 0x09, 0x06 }, // P
{ 0x3E, 0x41, 0x51, 0x21, 0x5E }, // Q
{ 0x7F, 0x09, 0x19, 0x29, 0x46 }, // R
{ 0x46, 0x49, 0x49, 0x49, 0x31 }, // S
{ 0x01, 0x01, 0x7F, 0x01, 0x01 }, // T
{ 0x3F, 0x40, 0x40, 0x40, 0x3F }, // U
{ 0x1F, 0x20, 0x40, 0x20, 0x1F }, // V
{ 0x3F, 0x40, 0x38, 0x40, 0x3F }, // W
{ 0x63, 0x14, 0x08, 0x14, 0x63 }, // X
{ 0x07, 0x08, 0x70, 0x08, 0x07 }, // Y
{ 0x61, 0x51, 0x49, 0x45, 0x43 }, // Z
{ 0x00, 0x7F, 0x41, 0x41, 0x00 }, // [
{ 0x55, 0x2A, 0x55, 0x2A, 0x55 }, // backslash
{ 0x00, 0x41, 0x41, 0x7F, 0x00 }, // ]
{ 0x04, 0x02, 0x01, 0x02, 0x04 }, // ^
{ 0x40, 0x40, 0x40, 0x40, 0x40 }, // _
{ 0x00, 0x01, 0x02, 0x04, 0x00 }, // '
{ 0x20, 0x54, 0x54, 0x54, 0x78 }, // a
{ 0x7F, 0x48, 0x44, 0x44, 0x38 }, // b
{ 0x38, 0x44, 0x44, 0x44, 0x20 }, // c
{ 0x38, 0x44, 0x44, 0x48, 0x7F }, // d
{ 0x38, 0x54, 0x54, 0x54, 0x18 }, // e
{ 0x08, 0x7E, 0x09, 0x01, 0x02 }, // f
{ 0x0C, 0x52, 0x52, 0x52, 0x3E }, // g
{ 0x7F, 0x08, 0x04, 0x04, 0x78 }, // h
{ 0x00, 0x44, 0x7D, 0x40, 0x00 }, // i
{ 0x20, 0x40, 0x44, 0x3D, 0x00 }, // j
{ 0x7F, 0x10, 0x28, 0x44, 0x00 }, // k
{ 0x00, 0x41, 0x7F, 0x40, 0x00 }, // l
{ 0x7C, 0x04, 0x18, 0x04, 0x78 }, // m
{ 0x7C, 0x08, 0x04, 0x04, 0x78 }, // n
{ 0x38, 0x44, 0x44, 0x44, 0x38 }, // o
{ 0x7C, 0x14, 0x14, 0x14, 0x08 }, // p
{ 0x08, 0x14, 0x14, 0x18, 0x7C }, // q
{ 0x7C, 0x08, 0x04, 0x04, 0x08 }, // r
{ 0x48, 0x54, 0x54, 0x54, 0x20 }, // s
{ 0x04, 0x3F, 0x44, 0x40, 0x20 }, // t
{ 0x3C, 0x40, 0x40, 0x20, 0x7C }, // u
{ 0x1C, 0x20, 0x40, 0x20, 0x1C }, // v
{ 0x3C, 0x40, 0x30, 0x40, 0x3C }, // w
{ 0x44, 0x28, 0x10, 0x28, 0x44 }, // x
{ 0x0C, 0x50, 0x50, 0x50, 0x3C }, // y
```

```
    { 0x44, 0x64, 0x54, 0x4C, 0x44 }, // z
    { 0x08, 0x6C, 0x6A, 0x19, 0x08 }, // {
    { 0x30, 0x4E, 0x61, 0x4E, 0x30 }, // |
    { 0x7E, 0x7E, 0x7E, 0x7E, 0x7E } // }
};


/*
* Standard ASCII characters in a 3x5 pixel font
*Numbers and upper case only. Characters 32-94
*/
static const unsigned char SmallFontLookup [][5] =
{
    { 0x00, 0x00, 0x00, 0xff, 0xff }, // sp
    { 0x17, 0xff, 0xff, 0xff, 0xff }, // !
    { 0x03, 0x00, 0x03, 0xff, 0xff }, // "
    { 0x0a, 0x1f, 0x0a, 0x1f, 0x0a }, // #
    { 0x1f, 0x1f, 0x1f, 0xff, 0xff }, // $
    { 0x13, 0x0b, 0x08, 0x1a, 0x19 }, // %
    { 0x1a, 0x15, 0x19, 0x14, 0xff }, // &
    { 0x03, 0xff, 0xff, 0xff, 0xff }, // '
    { 0x0e, 0x11, 0xff, 0xff, 0xff }, // (
    { 0x11, 0x0e, 0xff, 0xff, 0xff }, // )
    { 0x0a, 0x04, 0x0a, 0xff, 0xff }, // *
    { 0x04, 0x0e, 0x04, 0xff, 0xff }, // +
    { 0x10, 0x0c, 0xff, 0xff, 0xff }, // ,
    { 0x04, 0x04, 0x04, 0xff, 0xff }, // -
    { 0x10, 0xff, 0xff, 0xff, 0xff }, // .
    { 0x18, 0x0c, 0x06, 0x03, 0xff }, // /
    { 0x0e, 0x11, 0x0e, 0xff, 0xff }, // 0
    { 0x12, 0x1f, 0x10, 0xff, 0xff }, // 1
    { 0x19, 0x15, 0x12, 0xff, 0xff }, // 2
    { 0x15, 0x15, 0x0e, 0xff, 0xff }, // 3
    { 0x0f, 0x1c, 0x08, 0xff, 0xff }, // 4
    { 0x17, 0x15, 0x09, 0xff, 0xff }, // 5
    { 0x0e, 0x15, 0x09, 0xff, 0xff }, // 6
    { 0x19, 0x05, 0x03, 0xff, 0xff }, // 7
    { 0x0a, 0x15, 0x0a, 0xff, 0xff }, // 8
    { 0x02, 0x15, 0x0e, 0xff, 0xff }, // 9
    { 0x0a, 0xff, 0xff, 0xff, 0xff }, // :
    { 0x10, 0x0a, 0xff, 0xff, 0xff }, // ;
    { 0x08, 0x0a, 0x11, 0xff, 0xff }, // <
    { 0x0a, 0x0a, 0x0a, 0xff, 0xff }, // =
    { 0x11, 0x0a, 0x08, 0xff, 0xff }, // >
    { 0x01, 0x15, 0x02, 0xff, 0xff }, // ?
    { 0x0a, 0x15, 0x1d, 0x11, 0x0e }, // @
    { 0x1e, 0x05, 0x1e, 0xff, 0xff }, // A
    { 0x1f, 0x15, 0x0a, 0xff, 0xff }, // B
    { 0x0e, 0x11, 0x0a, 0xff, 0xff }, // C
    { 0x1f, 0x11, 0x0e, 0xff, 0xff }, // D
    { 0x1f, 0x15, 0x11, 0xff, 0xff }, // E
    { 0x1f, 0x05, 0x01, 0xff, 0xff }, // F
    { 0x0e, 0x11, 0x1d, 0xff, 0xff }, // G
    { 0x1f, 0x04, 0x1f, 0xff, 0xff }, // H
    { 0x11, 0x1f, 0x11, 0xff, 0xff }, // I
```

```
    { 0x04, 0x10, 0x0f, 0xff, 0xff }, // J
    { 0x1f, 0x04, 0x0a, 0x11, 0xff }, // K
    { 0x1f, 0x10, 0x10, 0xff, 0xff }, // L
    { 0x1f, 0x02, 0x04, 0x02, 0x1f }, // M
    { 0x1f, 0x06, 0x0c, 0x1f, 0xff }, // N
    { 0x0e, 0x11, 0x11, 0x0e, 0xff }, // O
    { 0x1f, 0x05, 0x02, 0xff, 0xff }, // P
    { 0x0e, 0x11, 0x19, 0x1e, 0xff }, // Q
    { 0x1f, 0x05, 0x1a, 0xff, 0xff }, // R
    { 0x12, 0x15, 0x09, 0xff, 0xff }, // S
    { 0x01, 0x1f, 0x01, 0xff, 0xff }, // T
    { 0x0f, 0x10, 0x10, 0x0f, 0xff }, // U
    { 0x07, 0x08, 0x10, 0x08, 0x07 }, // V
    { 0x0f, 0x19, 0x0c, 0x19, 0x0f }, // W
    { 0x1b, 0x04, 0x1b, 0xff, 0xff }, // X
    { 0x03, 0x1c, 0x03, 0xff, 0xff }, // Y
    { 0x19, 0x15, 0x13, 0xff, 0xff }, // Z
    { 0x1f, 0x11, 0xff, 0xff, 0xff }, // [
    { 0x02, 0x05, 0x02, 0xff, 0xff }, // backslash
    { 0x11, 0x1f, 0xff, 0xff, 0xff }, // ]
    { 0x02, 0x01, 0x02, 0xff, 0xff } // ^
};

#endif //miniEC_pcd8544_H
```

A.2.11 LCD module

Listing A.13: LCD module

```
#ifndef miniEC_pcd8544_C
#define miniEC_pcd8544_C

/* ============================================================*/
// This LCD will display 84x48 monochrome graphics or 14x6 characters of text.
//It uses a SPI-type interface to a micro.
//* -------- Improvements to make
  // - Don't refresh whole screen, update the screen with only the elements
  //that have changed.
/* ============================================================*/

#include <io.h>
#include <iomacros.h>
#include "hardware.h"
#include "pcd8544.h"


/*****************************************************************************
* Function name : lcd_Clear
* returns : none
* arguments : none
* Description : Clears the LCD's display memory
* Notes : lcd_Update must be called next
*****************************************************************************/
void lcd_clear(void) {

  static int i;

  for (i=0; i<LCD_CACHE_SIZE; i++)
  {
     LcdMemory[i] = 0;
  }
}


void lcd_clear_line(unsigned char row) {
  unsigned int i;
  unsigned int idx = (row*48*14)/8 ;

  for(i=0;i<84;i++){
    LcdMemory[idx] = 0;
    idx++;
  }
}

/*****************************************************************************
* Function name : lcd_update
* returns : none
* arguments : none
* Description : update the lcd screen after modification
* Notes  :
```

```
*****************************************************************************/

void lcd_update ( void )
{
  int i;

  // Set base address to X=0, Y=0
  lcd_send(0x80, SEND_CMD );
  lcd_send(0x40, SEND_CMD );

  // Send contents of memory buffer to LCD
  for (i=0; i<LCD_CACHE_SIZE; i++) {
    lcd_send( LcdMemory[i], SEND_CHR );
  }
}

/*****************************************************************************
* Function name : lcd_send
* returns : none
* arg1 : character to send
* arg2 : ID for data or command (0, or 1 respectively)
* Description :
* Notes  :
*****************************************************************************/

void lcd_send(unsigned char data, unsigned char cd) {

  // Enable display controller (active low).
  port3.out.pin0 = 0; // STE0 (CE)

  // Check for command or data
  if(cd == SEND_CHR) {
    port3.out.pin2 = 1; // SOMI0 (D/S)
  }
  else {
    port3.out.pin2 = 0; // SOMI0 (D/S)
  }

  //SEND via SPI
  U0TXBUF = data; //send data
  while((U0TCTL & TXEPT) == 0); //Wait for ready U0TXBUF
  port3.out.pin0 = 1; // Disable display controller STE0 (CE)
}


/*****************************************************************************
* Function name : lcd_init
* returns : none
* args : none
* Description : MCU SPI and LCD controller initialization.
* Notes  :
*****************************************************************************/

void lcd_init(void){
   // Pull-up on LCD reset pin.
```

```
  LCD_RES = 1;

  // Setup MCU ports for SPI connection
  P3DIR |= 0x0F; // P3.0,1,2,3 output direction
  P4DIR |= 0x02; // P4.1 res
  P3SEL |= 0x0A; // P3.1,3 SPI option select


  // Toggle LCD reset pin.
  LCD_RES = 0;
  delay(10000);
  LCD_RES = 1;

  // Init SPI
  U0CTL = 0x16; // SPI Mode, 8-bit, Master mode
  U0TCTL = 0xB2; // 3-wire Mode, clock->SMCLK, no CKPL (polarity), no CKPH (phase)
  U0BR0 = 0x02;
  U0BR1 = 0x00;
  UMCTL0 = 0x00; //SPI mode
  ME1 = 0x40; // Enable SPI0
  ME2 = 0x01; // Enable SPI0

  // Disable LCD controller.
  LCD_CS = 1;
  delay(100);

  // Send sequence of commands
  lcd_send( 0x21, SEND_CMD ); // Switch to LCD Extended Commands.
  lcd_send( 0xC8, SEND_CMD ); // Set LCD Vop (Contrast).
  lcd_send( 0x06, SEND_CMD ); // Set Temp coefficent.
  lcd_send( 0x13, SEND_CMD ); // LCD bias mode 1:48.
  lcd_send( 0x20, SEND_CMD ); // Switch back to LCD Standard Commands.
  lcd_send( 0x08, SEND_CMD ); // LCD blank
  lcd_send( 0x0C, SEND_CMD ); // LCD in normal mode.

  // Clear and Update
  lcd_clear();
  lcd_update();
}


/***********************************************************************
* Function name : lcd_write_Char
* returns : none
* arg1 : x coordinate, range 0 - 83
* arg2 : y coordinate, range 0 - 47
* arg3 : character
* Description : Displays a character at position x, y on the LCD
* Notes  : basic 5x7 1x font used
***********************************************************************/

void lcd_write_char(unsigned char x, unsigned char y, unsigned char ch)
{
    unsigned int idx = 0;
    unsigned int i = 0;
```

```
    // check for out of range positions
    if ( x > LCD_X_RES ) return;
    if ( y > LCD_Y_RES ) return;

    idx = (x*48 + y*48*14)/8 ;

    for ( i = 0; i < 5; i++ )
    {
      LcdMemory[idx] = FontLookup[ch - 32][i] << 1;
      idx++;
    }
}

/*****************************************************************************
* Function name : lcd_write_small_char
* returns : none
* arg1 : x coordinate, range 0 - 21
* arg2 : y coordinate, range 0 - 5
* arg3 : character
* Description : Displays a character at position x, y on the LCD
* Notes  : small 3x5 font
*****************************************************************************/

void lcd_write_small_char(unsigned char x, unsigned char y, unsigned char ch) {
    unsigned int idx = 0;
  unsigned int i = 0;
  unsigned char c = ch - 32;

  // check for out of range positions
  if(ch > 96){c = c - 32;}
  if ( x > LCD_X_RES ) return;
  if ( y > LCD_Y_RES ) return;

  idx = x*4 + y*84;
   //idx = (x*48 + y*48*21)/8 ;

    for(i = 0; i < 5; i++)
    {
        if (SmallFontLookup[c][i] == 0xff) break;
        LcdMemory[idx] = SmallFontLookup[c][i]<< 2;
        idx++;
        }
}

/*****************************************************************************
* Function name : lcd_write_2X_char
* returns : none
* arg1 : x coordinate, range 0 - 83
* arg2 : y coordinate, range 0 - 47
* arg3 : character
* Description : Displays a character at position x, y on the LCD
* Notes  : 2X font 10x14
*****************************************************************************/
```

```
void lcd_write_2X_char(unsigned char x, unsigned char y, unsigned char ch){
  unsigned int idx = 0;
  unsigned int i = 0;
  unsigned char temp0, temp1, temp2;

 // check for out off range
  if ( x > LCD_X_RES ) return;
  if ( y > LCD_Y_RES ) return;

  idx = (x*48 + y*48*14)/8 ;

 for (i = 0; i < 5; i++){
  temp0 = FontLookup[ch - 32][i] << 1;
  temp1 = (temp0 & 0x01) * 3;
  temp1 |= (temp0 & 0x02) * 6;
  temp1 |= (temp0 & 0x04) * 12;
  temp1 |= (temp0 & 0x08) * 24;

  temp0 >>= 4;
  temp2 = (temp0 & 0x01) * 3;
  temp2 |= (temp0 & 0x02) * 6;
  temp2 |= (temp0 & 0x04) * 12;
  temp2 |= (temp0 & 0x08) * 24;

  LcdMemory[idx++] = temp1;
  LcdMemory[idx++] = temp1;
  LcdMemory[idx + 82] = temp2;
  LcdMemory[idx + 83] = temp2;
  }

 // Update x cursor position.
  LcdMemIdx += 12;
}

/*****************************************************************************
* Function name : lcd_write_string
* returns : none
* arg1 : col coordinate, range 0 - 13
* arg2 : row coordinate, range 0 - 5
* arg2 : pointer to text
* Description : Displays a string on starting at position col, row
* Notes  : basic 5x7 1x font, no wrapping
*****************************************************************************/

void lcd_write_string(unsigned char col, unsigned char row, unsigned char *dataPtr)
{
  // loop to the end of string
  while ( *dataPtr ) {
    lcd_write_char(col, row, (*dataPtr));
    col++;
    dataPtr++;
  }
}

/*****************************************************************************
```

```
* Function name : lcd_small_string
* returns : none
* arg1 : col coordinate, range 0 - 20
* arg2 : row coordinate, range 0 - 7
* arg2 : pointer to text
* Description : Displays a string on starting at position col, row
* Notes  : basic 3x5 font
*******************************************************************************/

void lcd_write_small_string(unsigned char col, unsigned char row, unsigned char *dataPtr) {
  // loop to the end of string
    while(*dataPtr){
        lcd_write_small_char(col, row, (*dataPtr));
        col++;
    dataPtr++;
    }
}

/*******************************************************************************
* Function name : lcd_write_2X_string
* returns : none
* arg1 : col coordinate, range 0 - 13
* arg2 : row coordinate, range 0 - 5
* arg2 : pointer to text
* Description : Displays a string on starting at position col, row
* Notes  : 10x14 1x font,no wrapping
*******************************************************************************/

void lcd_write_2X_string(unsigned char col, unsigned char row, unsigned char *dataPtr)
{
  // loop to the end of string
  while ( *dataPtr ) {
    lcd_write_2X_char(col, row, (*dataPtr));
    col = col+2;
    dataPtr++;
  }
}

/*******************************************************************************
* Function name : lcd_draw_pixel
* returns : none
* arg1 : x coordinate, range 1 - 84
* arg2 : y coordinate, range 1 - 48
* arg3 : mode, options on, off or xor
* Description : Set, clear or invert a pixel at location x,y
* Notes  :
*******************************************************************************/

void lcd_set_pixel(unsigned char x, unsigned char y, unsigned char mode){
    unsigned int idx = 0;
    unsigned char offset = 0;
    unsigned char data = 0;

    // check for out off range
    if ( x > LCD_X_RES ) return;
```

```
    if ( y > LCD_Y_RES ) return;

    idx = ((y / 8) * 84) + x;
    offset = y - ((y / 8) * 8);

    data = LcdMemory[idx];

    if ( mode == PIXEL_OFF )
    {
        data &= (~(0x01 << offset));
    }
    else if ( mode == PIXEL_ON )
    {
        data |= (0x01 << offset);
    }
    else if ( mode == PIXEL_XOR )
    {
        data ^= (0x01 << offset);
    }

    LcdMemory[idx] = data;
}


/************************************************************************
* Function name : lcd_set_xy
* returns : none
* arg1 : x coordinate, range 0 - 13
* arg2 : y coordinate, range 0 - 5
* Description : Sets cursor location to xy location
* Notes  : 1X 5x7 font
************************************************************************/

void lcd_set_xy (unsigned char x, unsigned char y )
{
    LcdMemIdx = x*6 + y*84;
}

/************************************************************************
* Function name : lcd_contrast
* returns : none
* arg : contrast, range
* Description : Set LCD Contrast
* Notes  :
************************************************************************/

void lcd_set_contrast(unsigned char contrast) {

    // LCD Extended Commands.
    lcd_send( 0x21, SEND_CMD );

    // Set LCD Vop (Contrast).
    lcd_send( 0x80 | contrast, SEND_CMD );

    // LCD Standard Commands, horizontal addressing mode.
```

```
    lcd_send( 0x20, SEND_CMD );
}


/*****************************************************************************
* Function name : lcd_fill_image
* returns : none
* arg : pointer to image array
* Description : Draw bitmap image on LCD using the whole area
* Notes  :
*****************************************************************************/

void lcd_fill_image(const unsigned char* bitmap)
{
  static unsigned int i;

  lcd_clear();
  lcd_update();

  for (i=0; i<LCD_CACHE_SIZE; i++){lcd_send(bitmap[i],SEND_CHR);}

 /* for (i=0; i<LCD_Y_RES/8; i++){
    lcd_set_xy(0,i);

    for (j=0; j<LCD_X_RES; j++){
      lcd_send(bitmap[j+i*LCD_X_RES],SEND_CHR);
    }
  }
  */

  lcd_set_xy(0,0);
  delay(100);
}

/*-----------------------------------------------------------
  BORROWED CODE
  Credits: Sylvain Bissonnette's page at http://www.microsyl.com/
  Name : LcdLine

  Description : Draws a line between two points on the display.

  Argument(s) : x1, y1 -> Absolute pixel coordinates for line origin.
                x2, y2 -> Absolute pixel coordinates for line end.
                mode -> Off, On or Xor. See enum.

  Return value : None.

-----------------------------------------------------------*/

void LcdLine (unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2, unsigned char ↩
    mode )
{
    int dx, dy, stepx, stepy, fraction;

    dy = y2 - y1;
```

```
dx = x2 − x1;

if ( dy < 0 )
{
    dy = −dy;
    stepy = −1;
}
else
{
    stepy = 1;
}

if ( dx < 0 )
{
    dx = −dx;
    stepx = −1;
}
else
{
    stepx = 1;
}

dx <<= 1;
dy <<= 1;

lcd_set_pixel( x1, y1, mode );

if ( dx > dy )
{
    fraction = dy − (dx >> 1);
    while ( x1 != x2 )
    {
        if ( fraction >= 0 )
        {
            y1 += stepy;
            fraction −= dx;
        }
        x1 += stepx;
        fraction += dy;
        lcd_set_pixel( x1, y1, mode );
    }
}
else
{
    fraction = dx − (dy >> 1);
    while ( y1 != y2 )
    {
        if ( fraction >= 0 )
        {
            x1 += stepx;
            fraction −= dy;
        }
        y1 += stepy;
        fraction += dx;
        lcd_set_pixel( x1, y1, mode );
```

```
        }
    }
}

//LcdLine wrapped to follow this file's naming conventions
void lcd_draw_line(unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2, unsigned char ↩
     mode ){
  LcdLine(x1,y1,x2,y2,mode);
}

/*****************************************************************************
* Function name : lcd_draw_box
* returns : none
* arg1, arg2 : x1, y1, pixel coordinates for box origin. range 0,0...83,47
* arg1, arg2 : x2, y2, pixel coordinates for box end. range 0,0...83,47
* arg5 : mode, OFF,ON,XOR (outline,filled,inverted)
* Description : Draw a box outline, filled box or invert pixels in box area
* Notes   :
*****************************************************************************/
void lcd_draw_box(unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2, unsigned char ↩
     mode)
{
  unsigned char i,j;

  switch (mode){
  case(BOX_OFF): //box outline
    for (j=x1;j<=x2;j++){
      lcd_set_pixel(j,y1,PIXEL_ON);
      lcd_set_pixel(j,y2,PIXEL_ON);
    }
    for (i=y1;i<=y2;i++){
      lcd_set_pixel(x1,i,PIXEL_ON);
      lcd_set_pixel(x2,i,PIXEL_ON);
    }
    break;

  case(BOX_ON): //filled box
    for (i=y1;i<=y2;i++){
      for (j=x1;j<=x2;j++){
        lcd_set_pixel(j,i,PIXEL_ON);
      }
    }
    break;

  case(BOX_XOR): //inverted (pixels) box
    for (i=y1;i<=y2;i++){
      for (j=x1;j<=x2;j++){
        lcd_set_pixel(j,i,PIXEL_XOR);
      }
    }
    break;

  default : break;
  }
}
```

```
/*****************************************************************************
* Function name : lcd_clear_box
* returns : none
* arg1, arg2 : x1, y1, pixel coordinates for box origin. range 0,0...83,47
* arg1, arg2 : x2, y2, pixel coordinates for box end. range 0,0...83,47
* Description : set pixels defined by box to background
* Notes   :
*****************************************************************************/

void lcd_clear_box(unsigned char x1,unsigned char y1,unsigned char x2,unsigned char y2)
{
unsigned char i,j;

for (i=y1;i<=y2;i++){
  for (j=x1;j<=x2;j++){
    lcd_set_pixel(j,i,PIXEL_OFF);
    }
  }
}


#endif //miniEC_pcd8544_C
```

A.2.12 Interface to real time clock module

Listing A.14: Interface to real time clock module

```
#ifndef miniEC_rtc_H
#define miniEC_rtc_H

/*****************************************************************************
 *
 * rtc.h
 *
 *
 * No guarantees, warrantees, or promises, implied.
 *
 *
 *
 *****************************************************************************/

#include <io.h>
#include <iomacros.h>
#include <sys/inttypes.h>

#define YEAR1 2000
#define LEAP(year) (!((year)%4) && (((year)%100) || !((year)%400)))


//time structure
typedef struct {
  unsigned char hour, min, sec; //time
  unsigned char year, month, day; //date
}time_record;


//variables
extern time_record current_time; //current date and time
extern unsigned char *month_name[];

//function protoypes
void rtc_time_init(void);
void rtc_time_update(void);
void rtc_date_update(void);
void rtc_timerb_setup(void);
void rtc_time_display(void);
void rtc_time_string(unsigned char *buffer);


#endif //miniEC_rtc_H
```

A.2.13 Real time clock module

Listing A.15: Real time clock module

```
#ifndef miniEC_rtc_C
#define miniEC_rtc_C

/*****************************************************************************
 *
 * rtc.c
 *
 * This module provides rutines for a real time clock.
 *
 * No guarantees, warrantees, or promises, implied.
 *
 *****************************************************************************/

#include <io.h>
#include <signal.h>
#include <string.h>
#include <stdio.h>
#include "rtc.h"

//variables
//time_record current_time = {12,0,0,7,1,11}; //current date and time
unsigned char *month_name[]={"Jan","Feb","Mar","Apr","May","Jun",
                             "Jul","Aug","Sep","Oct","Nov","Dec"};


//interrupts
interrupt(TIMERB0_VECTOR) timerb_isr(void);

/*****************************************************************************
* get current time and date as a string
* format compliant with ISO 8601 numeric representation of dates and times
* hh:mm:ss (eg 12:20:30)
* Calendar date YYYY-MM-DD (eg 1997-07-16)
* "2002-05-07Thh:mm:ss.uuuuuuZ" %04d-%02d-%02dT%02d:%02d:%02d.%06dZ
*****************************************************************************/

void rtc_time_string(unsigned char *buffer){
  unsigned char *time_buffer;
  time_record now;

  time_buffer = buffer;
  now = current_time; //get current time

  // serialize date and time
  time_buffer += sprintf(time_buffer,"%04d-%02d-%02d %02d:%02d:%02d",
     now.year+YEAR1, now.month, now.day, now.hour, now.min, now.sec);
}


/*****************************************************************************
 * Function Name:
```

```
 ******************************************************************************/

void rtc_date_update(void){
if(++current_time.day == 32){ //count days up to 1 month
      current_time.day = 1; //reset days
    }
    else if (current_time.day == 31){ //for 30 day months
      if ((current_time.month == 4) || (current_time.month == 6) ||
          (current_time.month == 9) || (current_time.month == 11)){
        current_time.day=1;
       current_time.month++;
      }
    }
    //for month = feb and leap year
    else if((current_time.month == 2) && (current_time.day == 30)){
      current_time.day=1;
      current_time.month++;
    }
    //for month = Feb, not a leap year
    else if ((current_time.month == 2) && (current_time.day == 29) &&
      (!((current_time.year)%4) && ((current_time.year)%100))){
        //!(LEAP(current_time.year))){
      current_time.day=1;
      current_time.month++;
    }
    if (++current_time.month == 13){ //count months up to 1 year
      current_time.month = 1; //reset months and
      current_time.year++; //count years
    }
}


/******************************************************************************
 * Function Name:
 ******************************************************************************/

void rtc_timerb_setup(void){
  TBCTL = TBCLR + TBSSEL_ACLK; // clear counter, clock=ACLK
  TBCCTL0 = CCIE; // CCR0 interrupt enabled
  TBCCR0 = 32767; //32768 count = 1s
  //P1DIR |= 0x01; // P1.0 output
  TBCTL |= MC_1; // Start Timer_B in up mode
}

/*
void rtc_timera_setup(void){
  TACTL = TACLR + TASSEL_ACLK; //clear counter, clock = ACLK(32.768Hz)
  TACCTL0 = CCIE; //CCR0 interrupt enabled
  TACCR0 = 0x7FFF; //32768 count = 1s
  TACTL |= MC_1; //Start Timer_A in up mode
}
*/

/*----------------------------------
-Real time clock with timer interrupt
```

```
-------------------------------------*/
/*****************************************************************************
 * Function Name:
 *****************************************************************************/
interrupt(TIMERB0_VECTOR) timerb_isr(void){
  TBCCR0 += 32767; //reload CCR0
  if (++current_time.sec == 60){ //count seconds up to one minute
    current_time.sec = 0; //reset seconds
      if (++current_time.min == 60){ //count minutes up to one hour
        current_time. min = 0; //reset minutes
          if (++current_time.hour == 24){ //count hours up to one day
              current_time.hour = 0; //reset hours
           //rtc_date_update()
            if(++current_time.day == 32){ //count days up to 1 month
      current_time.day = 1; //reset days
    }
    else if (current_time.day == 31){ //for 30 day months
      if ((current_time.month == 4) || (current_time.month == 6) ||
          (current_time.month == 9) || (current_time.month == 11)){
        current_time.day=1;
      }
    }
    //for month = feb and leap year
    else if((current_time.month == 2) && (current_time.day == 30)){
      current_time.day=1;
    }
    //for month = Feb, not a leap year
    else if ((current_time.month == 2) && (current_time.day == 29) &&
      (!((current_time.year)%4) && ((current_time.year)%100))){
      current_time.day=1;
    }
    if (++current_time.month == 13){ //count months up to 1 year
      current_time.month = 1; //reset months and
      current_time.year++; //count years
    }

      }
    }
  }
 LPM3_EXIT;
}


#endif //miniEC_rtc_C
```

A.2.14 Interface to data storage module

Listing A.16: Interface to data storage module

```
#ifndef miniEC_storage_H
#define miniEC_storage_H


/*****************************************************************************
 *
 * $Workfile: storage.h $
 *
 * This module an interface to the data storage routines.
 *
 * No guarantees, warrantees, or promises, implied.
 *
 *****************************************************************************/

#include "globals.h"

extern unsigned char num_files;
extern unsigned int mem_left;
extern unsigned int mem_total;
extern unsigned int *data_pointer;

typedef struct {
  unsigned int marker1;
  unsigned int fileID, potential, interval, duration;
  unsigned int year, month, day; //date
  unsigned int hour, min, sec; //time
  unsigned int marker2;
} file_header;


typedef struct {
  unsigned int marker;
  unsigned int max, average;
} file_footer;

//prototype functions
unsigned int * find_next_location(void);
unsigned long get_mem_time(int ival);

void data_log(int mydata, char where);
void data_log_header(file_header head);
void data_log_tail(file_footer tail);
void get_data_log(void);
void data_log_value(int mydata[], short where);
void data_log_summary(void);
void data_log_read(void);
void data_log_erase(void);


#endif //miniEC_storage_H
```

A.2.15 Data storage module

Listing A.17: Data storage module

```
#ifndef miniEC_storage_C
#define miniEC_storage_C

/*****************************************************************************
 *
 * $Workfile: storage.c $
 *
 * This module provides routines for data storage and retrieval.
 *
 *
 *
 * No guarantees, warrantees, or promises, implied.
 *
 * Author:Sylvia Kwakye
 * Compiler: mspgcc
 * Target:MSP430F169
 *
 *****************************************************************************/

#include <io.h>
#include <iomacros.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#include "storage.h"
#include "hardware.h"

// variables
unsigned char num_files = 0; //number of files saved in memory
unsigned int *data_pointer = (unsigned int *)DATA_START;
unsigned int mem_total = (unsigned int)DATA_END - (unsigned int)DATA_START;
unsigned int mem_left = (unsigned int)DATA_END - (unsigned int)DATA_START;


/*-----------------------------------------
// duration in minutes
  -----------------------------------------*/

unsigned long get_mem_time(int ival){
  unsigned int head = sizeof(file_header);
  unsigned int foot = sizeof(file_footer);
  unsigned int dur = 0;


  dur = ((mem_left - head - foot)/sizeof(int))*ival; //seconds
  if(dur>60){dur = dur/60;} //minutes
  else {dur = 0;}

  return dur;
```

```
}


/*-----------------------------------------
// find_next_location
  -----------------------------------------*/
unsigned int * find_next_location(void){
  unsigned int *next_ptr = (unsigned int *)DATA_START;
  unsigned int *last_ptr = (unsigned int *)DATA_END;
  unsigned int counter = 0;
  num_files = 0;
  //mem_total = (unsigned int)last_ptr - (unsigned int)next_ptr;
  //mem_left = mem_total;


  while (next_ptr < last_ptr){
    //increment num_files when file marker is found
    if(*next_ptr == MARK1){num_files++;}

    //stop when finds an empty spot otherwise keep looking
    if((*next_ptr - 0xFFFF) == 0){
      mem_left = mem_left - (sizeof(int)*counter);
      //return next_ptr;
      break;
    }
    next_ptr++;
    counter++;
  }

  if(next_ptr == last_ptr){
    counter++;
    //lcd_write_string(0,1,"Out of Memory");
    mem_left = mem_left - counter*2;
  }

  return next_ptr;
}



/*-----------------------------------------
//assemble data file header metadata and write it to file
//void flash_write(void *dst, const void *src, unsigned int size);
//flash_write((void *)0x1000, &somestructure, sizeof(somestructure));
//highest write speed possible is 476khz
//each byte or word takes 33/Fftg seconds. eg. 33/455k = 73 microseconds
//Block programming is faster. Use that mode in next firmware version.
  -----------------------------------------*/
void data_log_header(file_header head){
  int size = sizeof(head);

  dint();
  //FCTL2 = FWKEY | FSSEL1 | 2; //clock = SMCLK, ~800k/2 => 400kHz
  //FCTL2 = FWKEY | FSSEL1 | 9; //clock = SMCLK, ~4M/9 => 455kHz
  if(mem_left > size){
```

```
    FCTL2 = FWKEY | FSSEL1 | 17; //clock = SMCLK, 8M, 8M/17 => 462607 Hz
    flash_write(data_pointer, &head, size);
  }
  data_pointer = data_pointer + (size/sizeof(int));
  mem_left -= size;
  eint();
}

/*-----------------------------------------
//assemble data file footer metadata if present and write it to file
-----------------------------------------*/
void data_log_tail(file_footer tail){
  int size = sizeof(tail);

  dint();
  if(mem_left > size){
    //FCTL2 = FWKEY | FSSEL1 | 2; //clock = SMCLK, ~800k/2 => 400kHz
    //FCTL2 = FWKEY | FSSEL1 | 9; //clock = SMCLK, ~4M/9 => 455kHz
    FCTL2 = FWKEY | FSSEL1 | 17; //clock = MCLK, 8M, 8M/17 => 462607 Hz
    flash_write(data_pointer, &tail, size);
  }

  data_pointer = data_pointer + (size/sizeof(int));
  mem_left -= size;
  eint();
}

/*-----------------------------------------
//if space available save data to flash
//hard coded the values of size for mydata. Remember to change if number
//of channels is changed.
  -----------------------------------------*/
void data_log_value(int *mydata, short where){

  unsigned char buffer[40];
  //volatile int i = 0;
  int size = 10; //5 int values in mydata = 10 bytes


  if(where == ON){
       sprintf(buffer, "%lu, %4d, %4d, %4d, %4d, %4d\n",
      //sprintf(buffer, "%lu %4d %4d %4d %4d %4d\n",
           dticks, mydata[0], mydata[1], mydata[2], mydata[3], mydata[4]);
      usart1_send_string(buffer);
    }

  else {

    if((5 + data_pointer) < (unsigned int *)DATA_END){
      //LED = ~LED;


      FCTL2 = FWKEY | FSSEL0 | 17; //clock = MCLK, 8M, 8M/17 => 462607 Hz
      dint();
      flash_write(data_pointer, mydata, size);
```

```
      eint();
      data_pointer += 5;
      mem_left -= size;

    }

    else {
      return;
    }
  }
}


/*-----------------------------------------
// data_log_summary
  -----------------------------------------*/
void data_log_summary(void){
  unsigned int *dataptr = (unsigned int *)DATA_START;
  //unsigned int fc = 0; //file counter
  unsigned char buffer1[35] = "";

  if((unsigned int)data_pointer == DATA_START){
    usart1_send_string("\n#Done: No files in memory\n");
    return;
  }

  //usart1_send_string("\n#Getting file headers\n");

  while (*dataptr != 0xFFFF && dataptr <= (unsigned int *)DATA_END){
    if(*dataptr == MARK1){
      //format header as file number [day, time, potential, duration, interval]
      sprintf(buffer1, "\n#Log %d [%d-%d-%d,%d:%d:%d,%d,%d,%d]",
          *(dataptr+1),
          *(dataptr+7),*(dataptr+6),*(dataptr+5),
          *(dataptr+8),*(dataptr+9),*(dataptr+10),
          *(dataptr+2),*(dataptr+4),*(dataptr+3));
      usart1_send_string(buffer1);

      dataptr = dataptr+12;
      //fc++;
      memset (buffer1, 0, 35);
    }
     //Now format and send data values
    else{
      dataptr++;
    }
  }
  usart1_send_string("#Done\n");
}



/*-----------------------------------------
//get all data available through usart
//compose header,
```

```
file_header metadata = {(unsigned int)MARK1,
                        num_files, potential, interval, duration,
                        (now.year+YEAR1), now.month, now.day,
                        now.hour, now.min, now.sec,
                        (unsigned int)MARK2};
  ----------------------------------------*/
void data_log_read(void){
  unsigned int *dataptr = (unsigned int *)DATA_START;
  //unsigned int fc = 0; //file counter
  unsigned int tc = 0; //interval time counter
  unsigned int ival = interval;
  unsigned char buffer1[35] = "";

  if((unsigned int)data_pointer == DATA_START){
    usart1_send_string("\n#Done:No files in memory\n");
    return;
  }

  usart1_send_string("\n#Getting all files\r\n");

  while (*dataptr != 0xFFFF && dataptr <= (unsigned int *)DATA_END){
    //if(*dataptr == MARK1 || *dataptr == MARK2 ){ usart1_send_string("\n#");}

    if(*dataptr == MARK1){
      //format header as file number [day, time, potential, duration, interval]
      sprintf(buffer1, "\n#Log %d [%d-%d-%d,%d:%d:%d,%d,%d,%d]",
          *(dataptr+1),
          *(dataptr+7),*(dataptr+6),*(dataptr+5),
          *(dataptr+8),*(dataptr+9),*(dataptr+10),
          *(dataptr+2),*(dataptr+4),*(dataptr+3));
      usart1_send_string(buffer1);

      //update the data pointer, set/reset file, interval counter and buffer
      ival = *(dataptr+3);
      dataptr = dataptr+12;
      //fc++;
      tc = 0;
      memset (buffer1, 0, 35);
    }

    //Now format and send data values
    else{
      sprintf(buffer1, "\n%d, %d, %d, %d, %d, %d",
          tc,*dataptr,*(dataptr+1),*(dataptr+2),*(dataptr+3),*(dataptr+4));
      usart1_send_string(buffer1);
      dataptr += 5;
      tc += ival;
    }
  }

  usart1_send_string("#Done\n");
}


/*----------------------------------------
```

```
//erase all data in memory
-----------------------------------------*/
void data_log_erase(void){
  if((unsigned int)data_pointer == DATA_START){
    usart1_send_string("#Done. No files in memory\n");
    return;
  }

  // usart1_send_string("#Erasing all files\n");
  //FCTL2 = FWKEY | FSSEL1 | 2; //clock = MCLK, ~800k/2 => 400kHz
  //FCTL2 = FWKEY | FSSEL1 | 9; //clock = MCLK, 4.096M/9 => 455111 Hz
  FCTL2 = FWKEY | FSSEL1 | 17; //clock = MCLK, 8M, 8M/17=> 462607 Hz

  flash_clear_memory();

  data_pointer = (unsigned int *)DATA_START;
  mem_total = (unsigned int)DATA_END - (unsigned int)DATA_START;
  mem_left = mem_total;
  num_files = 0;
  usart1_send_string("#Done. All files erased\n");
}

#endif //miniEC_storage_C
```

A.3 Software Application Listings

A.3.1 miniEC.py

```
#!/bin/env python
# -*- coding: iso-8859-15 -*-
#-----------------------------------------------------------------------
# Name: miniEC.py
# Author: Sylvia Kwakye
# Copyright: Sylvia Kwakye, BMB Labs, Cornell University
#-----------------------------------------------------------------------

import wx
import time
import wxmpl
import storage
from miniEC_wdr import *
from helpers import *
from driver import *
from plotting import *



# WDR: classes

class MyTextDialog(wx.Dialog):
    def __init__(self, parent, id, title,
        pos = wx.DefaultPosition, size = wx.DefaultSize,
        style = wx.DEFAULT_DIALOG_STYLE ):
        wx.Dialog.__init__(self, parent, id, title, pos, size, style)

        # WDR: dialog function TextDialogFunc for MyTextDialog
        TextDialogFunc( self, True )

        # WDR: handler declarations for MyTextDialog
        wx.EVT_BUTTON(self, wx.ID_OK, self.onClose)

    # WDR: methods for MyTextDialog

    def GetFileLoc(self):
        return self.FindWindowById( ID_FILE_LOC )

    def SetFileLoc(self, location):
        self.GetFileLoc().SetLabel(location)

    def GetTextctrlRecv(self):
        return self.FindWindowById( ID_TEXTCTRL_RECV )


    def DisplayFileData(self, datafile):
        try:
            f = file(datafile, 'r')
            self.GetTextctrlRecv().Clear()
```

```
            self.GetTextctrlRecv().SetValue(f.read())
            #self.SetTitle(datafile)
            f.close()
        except:
            dialog = wx.MessageDialog(self, "Error opening file","File Error", wx.OK|wx. ↩
                ICON_INFORMATION )
            dialog.CentreOnParent()
            dialog.ShowModal()
            dialog.Destroy()


    def UpdateTextView(self, x, newValues):
        #print newValues
        rtext = """\n%8d\t%6.2f\t%6.2f\t%6.2f\t%6.2f\t%6.2f"""%(x,newValues[0],newValues[1], ↩
            newValues[2],newValues[3],newValues[4])
        self.GetTextctrlRecv().AppendText(rtext)

    # WDR: handler implementations for MyTextDialog

    def onClose(self, event):
        #self.EndModal(wx.ID_OK)
        self.Close(True)



class MyEraseDialog(wx.Dialog):
    def __init__(self, parent, id, title,
        pos = wx.DefaultPosition, size = wx.DefaultSize,
        style = wx.DEFAULT_DIALOG_STYLE ):
        wx.Dialog.__init__(self, parent, id, title, pos, size, style)

        # WDR: dialog function EraseDialogFunc for MyEraseDialog
        EraseDialogFunc( self, True )

        # WDR: handler declarations for MyEraseDialog
        wx.EVT_BUTTON(self, wx.ID_CANCEL, self.onNo)
        wx.EVT_BUTTON(self, wx.ID_YES, self.onYes)

    # WDR: methods for MyEraseDialog

    # WDR: handler implementations for MyEraseDialog

    def onNo(self, event):
        return self.EndModal(wx.ID_CANCEL)


    def onYes(self, event):
        #-- call driver command to erase
        return self.EndModal(wx.ID_YES)


class MyDownloadDialog(wx.Dialog):
    def __init__(self, parent, id, title,
        pos = wx.DefaultPosition, size = wx.DefaultSize,
        style = wx.DEFAULT_DIALOG_STYLE ):
```

```
        wx.Dialog.__init__(self, parent, id, title, pos, size, style)


        self.parent = parent

        # WDR: dialog function DownloadDialogFunc for MyDownloadDialog
        DownloadDialogFunc( self, True )

        # WDR: handler declarations for MyDownloadDialog
        wx.EVT_BUTTON(self, wx.ID_OK, self.onDone)
        wx.EVT_BUTTON(self, ID_BUTTON_ERASE, self.onErase)
        wx.EVT_BUTTON(self, ID_BUTTON_DOWNLOAD, self.onDownload)


        self.SetTextComments("Querying miniEC. Please wait...")

        if parent.connected is False:
            self.SetTextctrlSummary("No miniEC device detected.\nMake sure a device is plugged in.")

        else:
            reply = parent.device.comm_get_data_headers()
            self.SetTextctrlSummary('')
            for x in reply:
                self.GetTextctrlSummary().AppendText(x + '\n')

        self.SetTextComments("0 files downloaded.")



    # WDR: methods for MyDownloadDialog

    def GetTextComments(self):
        return self.FindWindowById( ID_TEXT_COMMENTS )

    def SetTextComments(self, comment):
        self.GetTextComments().SetLabel(comment)

    def GetTextctrlSummary(self):
        return self.FindWindowById( ID_TEXTCTRL_SUMMARY )

    def SetTextctrlSummary(self, summary):
        self.GetTextctrlSummary().SetValue(summary)


    # WDR: handler implementations for MyDownloadDialog

    def onDone(self, event):
        self.EndModal(wx.ID_OK)


    def onErase(self, event):
        #-- verify that user wants to erase
        #-- update comments to show files erased
        #dialog = wx.MessageDialog(self, "X files erased","Erase", wx.OK|wx.ICON_INFORMATION )
        if self.parent.connected is False:
```

```
            self.parent.CreateMyDevice()

        if self.parent.connected is True:
            dialog = MyEraseDialog(self, -1, "MiniEC Meter Data")
            dialog.CentreOnParent()
            clicked = dialog.ShowModal()
            dialog.Destroy()

            if clicked == wx.ID_YES:
                self.SetTextComments("Querying miniEC. Please wait...")

                reply = self.parent.device.comm_clear_data()
                if reply.startswith('#Done'):
                    message = "Cleared MiniEC Meter memory"

                else:
                    message = "Could not clear memory. Retry"

                self.SetTextComments(message)



    def onDownload(self, event):
        #-- call driver command to get data
        #-- update comments to show files downloaded

        if self.parent.connected is False:
            parent.CreateMyDevice()

        if self.parent.connected is True:
            self.SetTextComments("Downloading from miniEC. Please wait...")

            reply = self.parent.device.comm_get_data()
            dtext = str(reply) + " files downloaded"

            dialog = wx.MessageDialog(self, dtext,"Download", wx.OK|wx.ICON_INFORMATION )
            dialog.CentreOnParent()
            dialog.ShowModal()
            dialog.Destroy()
            self.SetTextComments(dtext)


class MyStatusDialog(wx.Dialog):
    def __init__(self, parent, id, title, status,
        pos = wx.DefaultPosition, size = wx.DefaultSize,
        style = wx.DEFAULT_DIALOG_STYLE ):
        wx.Dialog.__init__(self, parent, id, title, pos, size, style)

        # WDR: dialog function StatusDialogFunc for MyStatusDialog
        StatusDialogFunc( self, True )

        #defaults
        #self.status = status
        self.GetTextStime().SetLabel(status.time)
        self.GetTextSdate().SetLabel(status.date)
```

```
        self.GetTextSpotential().SetLabel(str(status.potential))
        self.GetTextSduration().SetLabel(str(status.duration))
        self.GetTextSinterval().SetLabel(str(status.interval))
        self.GetTextSfiles().SetLabel(str(status.num_files))
        self.GetTextSmem().SetLabel(str(status.mem_total))
        self.GetTextSmemUsed().SetLabel(str(status.mem_used))
        self.GetTextSchan().SetLabel(str(status.num_channels))
        self.GetTextSsid().SetLabel(status.serial_id)
        self.GetTextSfirmware().SetLabel(status.firmware)

        # WDR: handler declarations for MyStatusDialog
        wx.EVT_BUTTON(self, wx.ID_CANCEL, self.onClose)



# WDR: methods for MyStatusDialog

def GetTextSfirmware(self):
    return self.FindWindowById( ID_TEXT_SFIRMWARE )

def GetTextSsid(self):
    return self.FindWindowById( ID_TEXT_sSID )

def GetTextSchan(self):
    return self.FindWindowById( ID_TEXT_sCHAN )

def GetTextSmemUsed(self):
    return self.FindWindowById( ID_TEXT_sMEM_USED )

def GetTextSmem(self):
    return self.FindWindowById( ID_TEXT_sMEM )

def GetTextSfiles(self):
    return self.FindWindowById( ID_TEXT_sFILES )

def GetTextSinterval(self):
    return self.FindWindowById( ID_TEXT_sINTERVAL )

def GetTextSduration(self):
    return self.FindWindowById( ID_TEXT_sDURATION )

def GetTextSpotential(self):
    return self.FindWindowById( ID_TEXT_sPOTENTIAL )

def GetTextSdate(self):
    return self.FindWindowById( ID_TEXT_sDATE )

def GetTextStime(self):
    return self.FindWindowById( ID_TEXT_sTIME )



# WDR: handler implementations for MyStatusDialog

def onClose(self, event):
```

```
        self.EndModal(wx.ID_CANCEL)


class MyOptionsDialog(wx.Dialog):
    def __init__(self, parent, id, title, settings,
        pos = wx.DefaultPosition, size = wx.DefaultSize,
        style = wx.DEFAULT_DIALOG_STYLE ):
        wx.Dialog.__init__(self, parent, id, title, pos, size, style)

        # WDR: dialog function OptionsDialogFunc for MyOptionsDialog
        OptionsDialogFunc( self, True )

        #defaults
        self.parent = parent
        self.settings = settings
        self.GetMyPotential().SetValue(settings.potential)
        self.GetMyDuration().SetValue(settings.duration)
        self.GetMyInterval().SetValue(settings.interval)
        self.range = 1250

        # WDR: handler declarations for MyOptionsDialog
        wx.EVT_BUTTON(self, wx.ID_CANCEL, self.onCancel)
        wx.EVT_BUTTON(self, wx.ID_OK, self.onUpload)

    # WDR: methods for MyOptionsDialog

    def GetCheckboxSync(self):
        return self.FindWindowById( ID_CHECKBOX_SYNC )

    def GetMyPotential(self):
        return self.FindWindowById( ID_MY_POTENTIAL )

    def SetMyPotential(self, newValue):
        #if(isnum(newValue)):
        # newValue = str(newValue)
        self.GetMyPotential().SetValue( str(newValue) )

    def GetMyDuration(self):
        return self.FindWindowById( ID_MY_DURATION )

    def SetMyDuration(self, newValue):
        #if(isnum(newValue)):
        # newValue = str(newValue)
        self.GetMyDuration().SetValue( str(newValue) )

    def GetMyInterval(self):
        return self.FindWindowById( ID_MY_INTERVAL )

    def SetMyInterval(self, newValue):
        #if(isnum(newValue)):
            #newValue = str(newValue)
        self.GetMyInterval().SetValue( str(newValue) )

    # WDR: handler implementations for MyOptionsDialog
```

```
    def onCancel(self, event):
        self.EndModal(wx.ID_CANCEL)

    def onUpload(self, event):
        if self.parent.connected is False:
            self.parent.CreateMyDevice()

        if self.parent.connected is True:
            self.settings.potential = self.GetMyPotential().GetValue()
            self.settings.interval = self.GetMyInterval().GetValue()
            self.settings.duration = self.GetMyDuration().GetValue()
            self.settings.range = 1250

            time_sync = self.GetCheckboxSync().GetValue()

            #--call driver command to upload new settings to mcu
            updated = self.parent.device.comm_update_options(self.settings.interval,
                                                             self.settings.duration,
                                                             self.settings.potential,time_sync)
            #--acknowledge upload with a dialog
            if updated is False:
                dialog = wx.MessageDialog(self, "MiniEC upload failed","Upload", wx.OK|wx. ↩
                    ICON_INFORMATION )
                dialog.CentreOnParent()
                dialog.ShowModal()
                dialog.Destroy()
            else:
                dialog = wx.MessageDialog(self, "MiniEC options uploaded","Upload", wx.OK|wx. ↩
                    ICON_INFORMATION )
                dialog.CentreOnParent()
                dialog.ShowModal()
                dialog.Destroy()

            self.EndModal(wx.ID_OK)


class MyPanelMeter(wx.Panel):
    def __init__(self, parent, id,
        pos = wx.DefaultPosition, size = wx.DefaultSize,
        style = wx.TAB_TRAVERSAL ):
        wx.Panel.__init__(self, parent, id, pos, size, style)

        self.myframe = parent

        # WDR: dialog function MeterPanelFunc for MyPanelMeter
        MeterPanelFunc( self, True )

        # WDR: handler declarations for MyPanelMeter
        wx.EVT_BUTTON(self, ID_BUTTON_STOP, self.onStopClick)
        wx.EVT_BUTTON(self, ID_BUTTON_START, self.onStartClick)

    # WDR: methods for MyPanelMeter

    def GetButtonStop(self):
```

```
    return self.FindWindowById( ID_BUTTON_STOP )

def GetButtonStart(self):
    return self.FindWindowById( ID_BUTTON_START )

def GetTextInfo(self):
    return self.FindWindowById( ID_TEXT_INFO )

def GetTextFile(self):
    return self.FindWindowById( ID_TEXT_FILE )

def SetTextFile(self, fileinfo):
    self.GetTextFile().SetLabel(fileinfo)

def GetCh5(self):
    return self.FindWindowById( ID_CH5 )

def SetCh5(self, newValue):
    if(isnum(newValue)):
        newValue = str(newValue)
    self.GetCh5().SetLabel( str(newValue) )

def GetCh4(self):
    return self.FindWindowById( ID_CH4 )

def SetCh4(self, newValue):
    if(isnum(newValue)):
        newValue = str(newValue)
    self.GetCh4().SetLabel( str(newValue) )

def GetCh3(self):
    return self.FindWindowById( ID_CH3 )

def SetCh3(self, newValue):
    if(isnum(newValue)):
        newValue = str(newValue)
    self.GetCh3().SetLabel( str(newValue) )

def GetCh2(self):
    return self.FindWindowById( ID_CH2 )

def SetCh2(self, newValue):
    if(isnum(newValue)):
        newValue = str(newValue)
    self.GetCh2().SetLabel( str(newValue) )

def GetCh1(self):
    return self.FindWindowById( ID_CH1 )

def SetCh1(self, newValue):
    if(isnum(newValue)):
        newValue = str(newValue)
    self.GetCh1().SetLabel( str(newValue) )

def SetAllCh(self, newValues):
```

```
            self.SetCh1('%6.2f' % newValues[0])
            self.SetCh2('%6.2f' % newValues[1])
            self.SetCh3('%6.2f' % newValues[2])
            self.SetCh4('%6.2f' % newValues[3])
            self.SetCh5('%6.2f' % newValues[4])


        # WDR: handler implementations for MyPanelMeter

    def onStopClick(self, event):
        self.GetButtonStop().Enable(False)
        stopped = self.myframe.device.comm_send_command(CMD_stop_run)
        self.myframe.stopIntervalTimer('#Stopped')

    def onStartClick(self, event):
        if self.myframe.connected is False:
            self.myframe.CreateMyDevice()

        if self.myframe.connected is True:
            self.GetButtonStop().Enable(True)
            self.GetButtonStart().Enable(False)
            ready = self.myframe.device.comm_send_command(CMD_start_run)
            #print "from gui, reply to start cmd is: %s" %ready
            run = self.myframe.device.comm_send_command(CMD_start_run)
            #print "from gui, reply to run cmd is: %s" %run
            self.myframe.initIntervalTimer()



class MyFrame(wx.Frame):
    def __init__(self, parent, id, title,
        pos = wx.DefaultPosition, size = wx.DefaultSize,
        style = wx.DEFAULT_FRAME_STYLE ):
        wx.Frame.__init__(self, parent, id, title, pos, size, style)

        try:
            self.SetIcon(wx.Icon("miniEC.ico", wx.BITMAP_TYPE_ICO))
        except:
            pass

        #print "initializing"
        self.CreateMyMenuBar()

        self.CreateMyToolBar()

        self.sb = self.CreateStatusBar(1)
        self.SetStatusText("Welcome!")
        #print "statusbar created"

        # initialize other variables
        self.running = None
        self.connected = False
        self.datafile = None
        self.filehandle = None
        self.dirname = os.getcwd()
```

```
        self.settings = miniEC_options()
        self.status = miniEC_status()
        self.duration = self.settings.duration*60/self.settings.interval
        self.interval = self.settings.interval
        self.numChannels = self.status.num_channels
        self.newValues = (0, 0, 0, 0, 0)
        self.dtick = 0


        # insert main window here
        self.mpanel = MyPanelMeter(self, -1)
        self.mpanel.GetButtonStop().Enable(False)
        self.mpanel.GetButtonStart().Enable(False)

        # initialize text view
        self.textview = MyTextDialog(self, -1, "MiniEC Monitor Text View" )
        self.plotview = MyPlotFigure(self, -1, "MiniEC Monitor Graph View")

        # WDR: handler declarations for MyFrame
        wx.EVT_MENU(self, ID_MENU_NEW, self.onNew)
        wx.EVT_MENU(self, ID_MENU_OPEN, self.onOpen)
        wx.EVT_MENU(self, ID_MENU_SAVEAS, self.onSave)
        wx.EVT_MENU(self, wx.ID_ABOUT, self.OnAbout)
        wx.EVT_MENU(self, wx.ID_EXIT, self.OnQuit)
        wx.EVT_CLOSE(self, self.OnCloseWindow)

        wx.EVT_MENU(self, ID_MENU_STATUS, self.onStatusSelect)
        wx.EVT_MENU(self, ID_MENU_DOWNLOAD, self.onDownloadSelect)
        wx.EVT_MENU(self, ID_MENU_UPLOAD, self.onUploadSelect)

        wx.EVT_MENU(self, ID_MENU_TEXT, self.onTextSelect)
        wx.EVT_MENU(self, ID_MENU_GRAPH, self.onGraphSelect)

        # date timer handler
        self.datetimer = wx.PyTimer(self.Notify)
        self.datetimer.Start(30000)
        self.Notify()

        # interval timer handler
        self.intervaltimer = wx.PyTimer(self.OnIntervalTimer)

        #initialize a miniEC device
        self.CreateMyDevice()


    # WDR: methods for MyFrame

    def CreateMyMenuBar(self):
        self.mb = MyMenuBarFunc()
        self.SetMenuBar( self.mb )
        #self.SetMenuBar( MyMenuBarFunc() )
        #print "menubar created"

    def CreateMyToolBar(self):
        self.tb = self.CreateToolBar(wx.TB_HORIZONTAL|wx.NO_BORDER)
```

```
    MyToolBarFunc( self.tb )
    #print "Toolbar created"


# instantiate a miniEC device
def CreateMyDevice(self):
    self.device = miniEC_comm()

# check device connection
    if not self.device.portname:
        self.connected = False
        alert = wx.MessageDialog(self, "No miniEC device detected.\nMake sure a device is plugged in. ↩
            ",
                                 "Alert: Missing Device", wx.OK|wx.ICON_EXCLAMATION )
        alert.CentreOnParent()
        response = alert.ShowModal()
        alert.Destroy()

    else:
        self.device.comm_open()
        self.device.comm_get_status()
        self.status = self.device.status
        self.settings = self.device.options
        self.connected = True

# WDR: handler implementations for MyFrame

def OnAbout(self, event):
    msg = """ Welcome to the MiniEC Monitor 1.0:

     * View the status of a miniEC device
     * Download stored data
     * Upload test settings
     * Run a test

    How To:

    * Selecting New from the File menu opens a new file and
      enables the start button. Clicking the START button
      starts an experiment.In this demo, the data is
      generated with simple functions. Click the STOP at any
      time during the experiment to stop the data generation.

    * View the data either in text or graph form by selecting
      View -> Text or View -> Graph from the View menu.

    * Open previously recorded files with File -> Open.

    * Save an opened file to another name using the Save as
      command under the File menu.

    * The stored data and status of a device can be
      downloaded and viewed by selecting the Data and
      Status items under the Device menu.
```

```
    * Device options can be updated with the Options dialog
      under the Device menu.

    """

    dialog = wx.MessageDialog(self, msg, "About", wx.OK|wx.ICON_INFORMATION )
    dialog.CentreOnParent()
    dialog.ShowModal()
    dialog.Destroy()


def OnQuit(self, event):
    #self.device.comm_close()
    self.Close(True)


def OnCloseWindow(self, event):
    if self.running is not None:
        alert = wx.MessageDialog(self, "Recording in progress, data will be lost.\nQuit anyway?",
                                 "Close MiniEC Monitor", wx.YES_NO|wx.ICON_QUESTION )
        alert.CentreOnParent()
        response = alert.ShowModal()
        alert.Destroy()

        if response == wx.ID_NO:
            pass

        if response == wx.ID_YES:
            self.datetimer.Stop()
            self.intervaltimer.Stop()
            #self.device.comm_close()
            self.Destroy()

    else:
        self.datetimer.Stop()
        self.intervaltimer.Stop()
        #self.device.comm_close()
        self.Destroy()


def onNew(self, event):
    datapath = None
    self.datafile = None
    filename = None

    dlg = wx.FileDialog(None, "Create new file...", self.dirname, 'test.dat', "Data File|*.dat|All Files|*", ↩
          wx.SAVE|wx.OVERWRITE_PROMPT)
    #dlg.SetFilename("test")
    if dlg.ShowModal() == wx.ID_OK:
        filename = dlg.GetFilename()
        self.datafile = dlg.GetPath()
        self.dirname=dlg.GetDirectory()
    dlg.Destroy()
```

```
    if self.datafile is not None:
        if filename.endswith(".dat") is False:
                filename = filename + '.dat'
                self.datafile = self.datafile + '.dat'

        self.mpanel.GetButtonStart().Enable(True)
        self.mpanel.GetTextFile().SetLabel(self.datafile)

        ## Forces a resize event to get TextFile label displayed properly
        self.SetSize(self.GetSize())

        """ Remove old windows """
        try:
          self.plotview.Destroy()
          self.textview.Destroy()
        except:
            pass

        self.filehandle = open(self.datafile,"w")
        self.filehandle.write("")
        self.filehandle.close()


        self.plotview = MyPlotFigure(self, -1, filename)
        self.textview = MyTextDialog(self, -1, filename)



def onOpen(self, event):
    self.datafile = None
    dlg = wx.FileDialog(None, "Open file...", self.dirname, "", "Data File|*.dat|All Files|*", wx.OPEN| ↩
          wx.FILE_MUST_EXIST)
    if dlg.ShowModal() == wx.ID_OK:
        self.datafile = dlg.GetPath()
        dlg.Destroy()

    if self.datafile is not None:
        """ If views have been deleted , create new ones"""
        try:
            self.textview.DisplayFileData(self.datafile)
        except:
            self.textview = MyTextDialog(self, -1, self.datafile)
            self.textview.DisplayFileData(self.datafile)

        try:
            self.plotview.PlotFileData(self.datafile)
        except:
            self.plotview = MyPlotFigure(self, -1, self.datafile)
            self.plotview.PlotFileData(self.datafile)

        self.plotview.Show()
        self.textview.Show()

        #self.mpanel.GetButtonStart().Enable(True)
        #self.mpanel.GetTextFile().SetLabel(self.datafile)
```

```
            self.mpanel.SetTextFile(self.datafile)


    def onSave(self, event):

        filename = self.datafile

        if self.datafile is not None:
            filename, postfix = os.path.splitext(filename)

            dlg = wx.FileDialog(None, "Save As...", self.dirname, filename, \
            "Data File|*.dat|XML File|*.xml", wx.SAVE|wx.OVERWRITE_PROMPT)

            if dlg.ShowModal() == wx.ID_OK:
                #filename = dlg.GetFilename()
                #self.dirname = dlg.GetDirectory()
                filename = dlg.GetPath()
                #ext = dlg.GetWildcard()
                #filename += ext;
                dlg.Destroy()

            if filename.endswith(".xml"):
                storage.text_to_xml(datafile, filename)

            else:
                self.textview.GetTextctrlRecv().SaveFile(filename)
                if filename.endswith(".dat") is False:
                    filename = filename + '.dat'

        else:
            dialog = wx.MessageDialog(self, "No data available to save",
            "Export Error", wx.OK|wx.ICON_INFORMATION )
            dialog.CentreOnParent()
            dialog.ShowModal()
            dialog.Destroy()


    def onStatusSelect(self, event):

        dialog = MyStatusDialog(self, -1, "MiniEC Meter Status", status=self.status )
        dialog.CentreOnParent()
        dialog.ShowModal()
        dialog.Destroy()


    def onDownloadSelect(self, event):
        dialog = MyDownloadDialog(self, -1, "MiniEC Meter Data")
        dialog.CentreOnParent()
        dialog.ShowModal()
        dialog.Destroy()


    def onUploadSelect(self, event):

        dialog = MyOptionsDialog(self, -1, "MiniEC Meter Options", settings=self.settings )
```

```
        dialog.CentreOnParent()
        result = dialog.ShowModal()
        dialog.Destroy()

        #quick debug
        #print "after"
        #s = self.status.list_miniEC_status()
        #print s
        #print "potential -- ", self.settings.potential
        #print "interval -- ", self.settings.interval
        #print "duration -- ", self.settings.duration
        #print "range -- ", self.settings.range


    def onTextSelect(self, event):
        self.textview.Show()


    def onGraphSelect(self, event):
        self.plotview.Show()


    # #----------------------------------------------------
    # SBK:additional functions not necessarily gui related
    # mostly timer
    #----------------------------------------------------
    #30-sec Time-out handler
    def Notify(self):
        t = time.localtime(time.time())
        st = time.strftime("%d-%b-%Y %I:%M %p", t)
        self.sb.SetStatusText(st, 0)


    def initIntervalTimer(self):
         self.running = 'running'
         self.mb.EnableTop(1, False)
         self.mb.EnableTop(0, False)
         self.dtick = 0
         self.newValues = (0, 0, 0, 0, 0)
         self.filehandle = open(self.datafile, 'a' )

         i = self.settings.interval
         d = self.settings.duration*60
         t = time.localtime(time.time())

         #Metadata list. The software version, user_id, test_label and note variables
         # are empty for now.
         meta = [self.settings.potential,self.settings.interval,self.settings.duration,\
                 self.settings.range,self.status.num_channels,self.status.serial_id,\
                 self.status.firmware,' ', time.strftime("%I:%M:%S", t), \
                 time.strftime("%d-%b-%Y", t),' ', ' ', ' ']

         rtext = storage.make_text_header(meta)
         self.textview.GetTextctrlRecv().AppendText(rtext)
         self.filehandle.write(rtext)
```

```
        self.plotview.init_plot_data(d, i)
        self.device.comm_flush()
        self.intervaltimer.Start(i*1000)
        #self.onGraphSelect(True)


def OnIntervalTimer(self):
    i = self.dtick*self.settings.interval

    data = self.device.comm_read_datapoint()
    print data
    while data.startswith('#'):
        if data.startswith('#stop') or data.startswith('#Done'):
            return
        else :
            data = self.device.comm_read_datapoint()


    values = map(int,data.split(',')) #split on comma
    #print values

    #self.newValues = readdata(self.dtick)
    self.newValues = values[1:]
    #print self.newValues[0]
    #print self.newValues[4]
    self.setElapsedTime()

    self.mpanel.SetAllCh(self.newValues)
    self.textview.UpdateTextView(i, self.newValues)
    self.plotview.UpdatePlotView(self.dtick, self.newValues)

    rtext = """\n%8d\t%6.2f\t%6.2f\t%6.2f\t%6.2f\t%6.2f"""%\
            (i,self.newValues[0],self.newValues[1],self.newValues[2],\
             self.newValues[3],self.newValues[4])
    self.filehandle.write(rtext)

    self.dtick = self.dtick + 1
    if self.dtick >= self.settings.duration*60/self.settings.interval:
        self.stopIntervalTimer()
        #last = self.device.comm_expt_stop()
        #print "from gui, last reading:%s" %last
        #self.device.comm_flush()


def stopIntervalTimer(self,message=None):
    self.intervaltimer.Stop()
    self.mpanel.GetButtonStop().Enable(False)
    if message is not None:
      self.textview.GetTextctrlRecv().AppendText('\n'+ message)
      self.filehandle.write('\n'+ message)
    #self.textview.GetTextctrlRecv().SaveFile(self.datafile)
    self.running = None
    self.filehandle.close()
```

```
        self.mb.EnableTop(1, True)
        self.mb.EnableTop(0, True)



    def setElapsedTime(self):
        """ Set the time string to Hours:Minutes:Seconds """
        t = self.dtick*self.settings.interval
        if(t <60):
            hours = 0
            minutes = 0
            seconds = t
        if(t < 3600):
            hours = 0
            minutes = int(t/60)
            seconds = t - minutes*60
        if(t >=3600):
            hours = int(t/3600)
            minutes = int((t - hours*3600)/60.0)
            seconds = t - hours*3600 - minutes*60

        tstr = """%02d:%02d:%02d""" %(hours, minutes, seconds)
        self.mpanel.GetTextInfo().SetLabel(tstr)


#-----------------------------------

class MyApp(wx.App):

    def OnInit(self):
        wx.InitAllImageHandlers()
        frame = MyFrame( None, -1, "MiniEC Monitor", [20,20], [400,400] )
        frame.Show(True)

        return True

#-----------------------------------

app = MyApp(True)
app.MainLoop()
```

A.3.2 driver.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------
# Name: driver.py
# Author: Sylvia Kwakye
# Copyright: Sylvia Kwakye, BMB Labs, Cornell University
#-------------------------------------------------------------------------

import sys
import serial
import threading
import os
import time
from helpers import *
#import usb
#import logging


#device descriptors
vendor_id = 0x0403
product_id = 0x6001
serial_id = "ECPT5H6J"
baudrate = 9600
bytesize = serial.EIGHTBITS
parity = serial.PARITY_NONE
stopbits = serial.STOPBITS_ONE

#commands
CMD_product_id = 'pi'
CMD_vendor_id = 'vi'
CMD_serial_id = 'si'
CMD_get_files = 'gf'
CMD_get_headers = 'gh'
CMD_get_status = 'gs'
CMD_get_time = 'gt'
CMD_clear_files = 'sc'
CMD_set_options = 'sp'
CMD_set_time = 'st'
CMD_start_run = 'mr'
CMD_stop_run = 'ms'

RESPONSE_OK = 'ok'
RESPONSE_NOT_VALID = 'Command not valid. Type hp for menu'



class miniEC_options :
    """Holder for DC amperometry settings and time that can be
    changed by the user

    """

    def __init__(self):
```

```
        self.potential = 400 #potential in millivolts
        self.interval = 1 #interval in seconds
        self.duration = 1 #duration in minutes
        self.range = 1250 #current range, fixed 0 - 1250 nA



    def set_options_data(self, p,i,d):
        self.potential = int(p)
        self.interval = int(i)
        self.duration = int(d)



class miniEC_status :
    """Holder for miniEC device status information."""

    def __init__(self):
        #default status values
        self.potential = 400
        self.interval = 1
        self.duration = 1
        self.range = 1250

        self.num_files = 0
        self.mem_total = 0
        self.mem_used = 0

        self.num_channels = 0
        self.serial_id = "******"
        self.firmware = "0.0"

        self.time = "00:00:00"
        self.date = "00-00-2000"



    def set_status_data(self, status_data):
        """Update status data from miniEC"""

        self.date = status_data[0]
        self.time = status_data[1]

        self.potential = int(status_data[2])
        self.duration = int(status_data[3])
        self.interval = int(status_data[4])
        self.range = 1250

        self.num_files = int(status_data[5])
        self.mem_total = int(status_data[6])
        self.mem_used = int(status_data[7])

        self.num_channels = int(status_data[8])
        self.serial_id = status_data[9]
        self.firmware = status_data[10]
```

```
    def list_miniEC_status(self):
        """return status as a list"""

        status_list = self.time, self.date, self.potential, self.duration, \
        self.interval,self.range,self.num_files,self.mem_total,self.mem_used,\
        self.num_channels,self.serial_id,self.firmware

        return status_list



class miniEC_comm(object) :
    """ miniEC_comm abstracts all communication with a miniEC device."""

    data_buffer = ''
    file_buffer = ''
    is_running = False


    def __init__(self) :

        #self._log = logging.getLogger('%s.%s' % (__name__, self.__class__.__name__))

        self.options = miniEC_options()
        self.status = miniEC_status()

        self.portname = self.comm_find_device()
        self.handle = None



    def comm_find_device(self):

        """
        Scan for all available ports. build a list of tuples containing the
        port number and the port name (num, port-name). USB devices are
        detected as virtual RS232 ports.
        """

        ports = []
        found = None

        for i in range(16):
            try:
                s = serial.Serial(i)
                ports.append( (i, s.portstr))
                s.close()

            except serial.SerialException:
                pass
```

```
    #If using a posix distribution, scan for USB as follows
    if os.name == "posix":
        for i in range(16):
            try:
                ser_name = '/dev/ttyUSB' + str(i)
                s = serial.Serial(ser_name)
                ports.append((i,s.portstr))
                s.close()

            except serial.SerialException:
                pass


    """
    Autodetect which port the miniEC is connected to by querying each
    port in the list for a hard-wired ID string. Used the serial ID here
    but one can use either the vendor or product IDS or combination of
    all the IDs.
    """

    for n, n_name in ports:
        dev = serial.Serial(n_name,timeout=1.1)
        #self._log.info(dev.portstr)
        print dev.portstr
        dev.flush()

        dev.write('si\r')
        s_id = dev.read(8)
        print s_id
        dev.flush()
        dev.close()

        if s_id == serial_id:#stick a condition here
            #self._log.debug("product id at %s: %s\n" % (n_name, s_id))
            #print "serial id at %s: %s\n" % (n_name, s_id)
            found = n_name
        else:
            pass
            #self._log.debug("miniEC not on port %s\n" % (n_name))
            #print "miniEC not on port %s\n" % (n_name)

    return found


def comm_open(self) :
    self.handle = serial.Serial(self.portname,
                        baudrate,
                        bytesize=bytesize,
                        parity=parity,
                        stopbits=stopbits,
                        timeout=0.5
                                )
```

```
def comm_close(self) :
    if self.handle:
        self.handle.close()
        #self.handle = None
    else:
        raise RunTimeError, 'Device not open'

def comm_flush(self) :
    if self.handle:
        self.handle.flushOutput()
        self.handle.flushOutput()
    else:
        raise RunTimeError, 'Device not open'



def comm_send_command(self, command) :
    """
    send a command to the miniEC.
    """
    self.comm_flush()
    cmd = command + '\r'
    self.handle.write(cmd)
    reply = ''

    #wait for a complete line
    while True:
        reply = reply + self.handle.read(self.handle.inWaiting())
        if reply.endswith('\n') is True:
            break

    #print "reply to %s is: %s" %(cmd, reply)

    return reply


def comm_read_datapoint(self) :
    """
    get a line of data from the miniEC.
    """

    data = ''

    #wait for a complete line
    while True:
        data = data + self.handle.read(self.handle.inWaiting())
        if data.endswith('\n') is True:
            break


    #print "this line reads: %s" %(data)
    data.strip() #remove extra spaces

    return data
```

```
def comm_get_data(self) :
    """
    Get stored file data from the miniEC.
    """
    downloaded = 0
    self.comm_flush()
    cmd = CMD_get_files + '\r'
    self.handle.write(cmd)

    while True:
        self.file_buffer = self.file_buffer + self.handle.read(self.handle.inWaiting())
        if '#Done' in self.file_buffer:
            break

    f = False

    for line in self.file_buffer.split('\n'):
        if "#Log" in line:
            l1 = line.split(" ") # split on space
            l2 = l1[2][1:-1] #remove braces
            l3 = l2.split(',')


            filename = "logged_" + l3[0] + '_' + l3[1] + ".dat"

            if file_exists(filename):
                filename = "logged_" + l3[0] + '_' + l3[1] + '_(2)' + ".dat"

                while file_exists(filename):
                    filename = increment(filename)


            newfile = open(filename, "w")
            newfile.close

            metatext= """#Experiment: DCPA\n#Potential: %d mV\n#Interval: %d sec\n#Duration
                : %d min\n#Range: %d nA\n#Time: %s"""\
             %(int(l3[2]),int(l3[4]),int(l3[3]),int(self.status.range),l3[0] + " " + l3[1])
            print >>newfile, metatext

            metatext = """#\n#%8s\t%6s\t%6s\t%6s\t%6s\t%6s\n#""" %("Time (s)","Chan-1"," Chan-2 ","Chan-3 ","Chan-4 ","Chan-5 ")
            print >>newfile, metatext

            f = True
            downloaded += 1
            print filename

        else:
            if f is True:
                line.strip()
                print >>newfile, line.replace(",","\t")

    #print self.file_buffer
```

```
        return downloaded


    def comm_get_data_headers(self) :
        """
        Get stored file data from the miniEC.
        """
        headers = ''
        reply = []
        self.comm_flush()
        cmd = CMD_get_headers + '\r'
        self.handle.write(cmd)


        while True:
            headers = headers + self.handle.read(self.handle.inWaiting())
            if '#Done' in headers:
                break

        for line in headers.split('\n'):
            if "#Log" in line:
                print line
                reply.append(line)

        self.comm_flush()
        return reply


    def comm_clear_data(self) :
        """
        Delete stored data on the miniEC.
        """
        self.comm_flush()
        cmd = CMD_clear_files
        #self.handle.write(cmd)
        #reply = self.handle.readline()
        reply = self.comm_send_command(cmd)

        print "reply to %s is: %s" %(cmd, reply)
        self.comm_flush()
        return reply


    def comm_get_status(self) :
        """
        Get status data from the miniEC.
        """
        self.comm_flush()
        cmd = CMD_get_status + '\r'
        self.handle.write(cmd)

        reply = self.handle.read(64)
        #reply = self.comm_send_command(cmd)
        reply = reply[1:-3] #remove braces and \r\n
```

```
    items = reply.split(',') #split on comma
    for item in items:item.strip() #remove whitespace

    self.status.set_status_data(items)
    self.options.set_options_data(self.status.potential,
                                  self.status.interval,
                                  self.status.duration)
    self.comm_flush()
    return reply


def comm_update_options(self, ival, dur, pot, sync_time) :



    """
    Set amperometry options on miniEC.
    """
    self.comm_flush()
    check = False

    if(sync_time is True):
        self.comm_set_time()
        self.comm_flush()

    print "Updating interval, potential and duration"
    cmd = CMD_set_options + " %d %d %d\r" % (ival, dur, pot)
    self.handle.write(cmd)
    reply = self.handle.read(64)
    #reply = self.comm_send_command(cmd)

    print "reply to %s is: %s" %(cmd, reply)

    #reply.rstrip('\r\n') #remove any combination of \r\n
    reply = reply[1:-3] #remove braces and \r\n
    items = reply.split(',') #split on comma
    for item in items:item.strip() #remove whitespace

    #check
    if pot == int(items[2]) and dur==int(items[3]) and ival == int(items[4]):
        check = True

    self.status.set_status_data(items)
    self.options.set_options_data(self.status.potential,
                                  self.status.interval,
                                  self.status.duration)

    self.comm_flush()
    return check



def comm_get_time(self) :
    """
```

```
    Get time from the miniEC.
    """

    self.comm_flush()
    cmd = CMD_get_time + '\r'
    self.handle.write(cmd)
    reply = self.handle.read(20)

    #print "reply to %s is: %s" %(cmd, reply)
    self.comm_flush()
    return reply


def comm_set_time(self) :
    """
    Synchronize clocks between PC and device
    """
    self.comm_flush()
    t = time.gmtime()
    print t
    cmd = CMD_set_time + " " + time.strftime("%d %m %Y %H %M %S", t) + '\r'
    print cmd
    self.handle.write(cmd)
    reply = self.handle.read(20)
    print reply

    print "reply to %s is: %s" %(cmd, reply)
    self.comm_flush()
    return reply



def comm_expt_start(self) :
    """
    1. write time and params to data file
    2. send the run command, (set running flag?)
    3. start the running thread to receive data
    """

    #print "In comm_expt_start"

    reply = self.comm_send_command(CMD_start_run)
    print "reply to start is: %s" %(reply)

    if reply.startswith("#Ready"):
        print "Leaving comm_expt_start"
        self.comm_expt_run()



def comm_expt_run(self) :
    """
    Serial --> buffer/file
    1. write each data point
    2. display each data point
```

```
        3. do until 'done' signal received
        4. call stop to end thread
        """

        print "In comm_expt_run"
        print "To stop data collection, type <CTRL> + c."

        self.comm_flush()
        cmd = CMD_start_run #run
        reply = self.comm_send_command(CMD_start_run)
        print "reply to run is: %s" %(reply)


        if reply.startswith("#run"):
            self.is_running = True


        while self.is_running is True:
            try :
                data = self.comm_read_datapoint()
                print data

                if data.startswith("#Done") is True or data.startswith("#stop")is True:
                    self.is_running = False


            except KeyboardInterrupt :
                print "Stopped data logging."
                self.comm_expt_stop()


        print "Leaving comm_expt_run"
        self.comm_flush()
        return



    def comm_expt_stop(self) :
        """
        Stop a running experiment
        calculate summary stats
        add stats to log file
        close data file
        """

        print "In comm_expt_stop"

        self.comm_flush()
        cmd = CMD_stop_run
        reply = self.comm_send_command(cmd)

        self.is_running = False

        print "reply to %s is: %s" %(cmd, reply)
        print "Leaving comm_expt_stop"
```

```
            self.comm_flush()
            return reply




if __name__=='__main__':
    device = miniEC_comm()

    if not device.portname:
        raise Exception, "No miniEC device detected"
    else:
        device.comm_open()
        #print device.portname

    print "getting data"
    device.comm_get_data()


    print reply

    device.comm_close()
```

A.3.3 plotting.py

```
#!/bin/env python
# -*- coding: iso-8859-15 -*-
#-----------------------------------------------------------------
# Name: plotting.py
# Author: Sylvia Kwakye
# Copyright: Sylvia Kwakye, BMB Labs, Cornell University
#-----------------------------------------------------------------

import wx
import time
import wxmpl
import matplotlib
import matplotlib.numerix as numpy
from matplotlib.pylab import *
from miniEC_wdr import *
from helpers import *
from driver import *
from matplotlib.figure import Figure
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg
from matplotlib.backends.backend_wxagg import NavigationToolbar2WxAgg
from matplotlib.backends.backend_wx import _load_bitmap



class TestChannel(wxmpl.Channel):
    """
    A data-provider that reveals another point every time its tick() method
    is called.
    """
    def __init__(self, name, x, y, marker=None):
        """
        Creates a new C TestChannel with the matplotlib label name, marker and X
        and Y vectors.
        """
        wxmpl.Channel.__init__(self, name)
        self.x = x
        self.y = y
        self.idx = 0
        self.marker = marker

    def getX(self):
        """
        Returns the current X vector.
        """
        return self.x[0:self.idx]

    def getY(self):
        """
        Returns the current Y vector.
        """
        return self.y[0:self.idx]

    def getMarker(self):
```

```
        """
        Returns the line marker to use.
        """
        return self.marker

    def tick(self):
        """
        Reveals another point from the source X and Y vectors.
        """
        if self.idx < self.x.shape[0]:
            self.setChanged(True)
            self.idx += 1



class MyCustomToolbar(NavigationToolbar2WxAgg):
    def __init__(self, canvas):
        # create the default toolbar
        NavigationToolbar2WxAgg.__init__(self, canvas)

        #ON_CUSTOM_PRINT = wx.NewId()

        # delete unwanted tools
        self.DeleteToolByPos(6) # subplots
        self.DeleteToolByPos(5) # separator
        self.DeleteToolByPos(4) # zoom
        self.DeleteToolByPos(3) # pan
        self.DeleteToolByPos(2) # right
        self.DeleteToolByPos(1) # left
        self.DeleteToolByPos(0) # home



class MyPlotFigure(wx.Dialog):
    def __init__(self, parent, id, title,
        pos = wx.DefaultPosition, size = wx.DefaultSize,
        style = wx.DEFAULT_DIALOG_STYLE | wx.RESIZE_BORDER ):
        wx.Dialog.__init__(self, parent, id, title, pos, size, style)

        #self.myframe = wx.GetTopLevelParent(self)

        self.numPoints = 60 #default = 60 second duration


        self.canvas = wxmpl.PlotPanel(self, -1)
        self.fig = self.canvas.get_figure()
        self.axes = self.fig.gca()
        self.axes.grid(True)
        self.axes.set_xlabel('Time (sec)')
        self.axes.set_ylabel('Current (nA)')
        #self.toolbar = NavigationToolbar2WxAgg(self.canvas)
        self.toolbar = MyCustomToolbar(self.canvas)
        self.toolbar.Realize()
```

```
        #put everything in a sizer
        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(self.canvas, 1, wx.LEFT|wx.TOP|wx.GROW)
        sizer.Add(self.toolbar, 0, wx.GROW)
        self.SetSizer(sizer)
        self.Fit()


    #methods for PlotFigure
    def init_plot_data(self,dur,ival):
        self.numPoints = dur
        self.x = numpy.arange(0.0,dur,ival)
        self.y1 = 0*self.x
        self.y2 = 0*self.x
        self.y3 = 0*self.x
        self.y4 = 0*self.x
        self.y5 = 0*self.x

        # Attach the StripCharter and define its channels
        self.charter = wxmpl.StripCharter(self.axes)
        self.charter.setChannels([
            TestChannel('ch1', self.x, self.y1, marker='d'),
            TestChannel('ch2', self.x, self.y2, marker='v'),
            TestChannel('ch3', self.x, self.y3, marker='s'),
            TestChannel('ch4', self.x, self.y4, marker='o'),
            TestChannel('ch5', self.x, self.y5, marker='x')
        ])
        self.canvas.draw()


    def GetToolBar(self):
        return self.toolbar


    def UpdatePlotView(self,idx, ys):
        self.y1[idx] = ys[0]
        self.y2[idx] = ys[1]
        self.y3[idx] = ys[2]
        self.y4[idx] = ys[3]
        self.y5[idx] = ys[4]

        #if self.numPoints == self.charter.channels[0].x.shape[0]:
        # self.myframe.stopIntervalTimer('#Stopped')

        for channel in self.charter.channels:
            channel.tick()

        self.charter.update()




    def PlotFileData(self, filename):
        #load the data from file
        self.Y = load(filename, comments='#')
```

```
        #plot data
        #self.axes = self.fig.add_subplot(111)
        self.axes.cla()
        self.axes.plot(self.Y[:,0],self.Y[:,1], 'd-',\
        self.Y[:,0],self.Y[:,2], 'v-',\
        self.Y[:,0],self.Y[:,3], 's-',\
        self.Y[:,0],self.Y[:,4], 'o-',\
        self.Y[:,0],self.Y[:,5], 'x-', markersize=5)
        self.axes.legend(('ch1', 'ch2', 'ch3','ch4','ch5'), loc='best', shadow=True)
        self.axes.grid(True)
        self.axes.set_xlabel('Time (sec)')
        self.axes.set_ylabel('Current (nA)')
        self.canvas.draw()
        self.SetTitle(filename)


    def onEraseBackground(self, evt):
        pass
```

A.3.4 storage.py

```
#!/usr/bin/env python
# −∗− coding: utf−8 −∗−
#−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
# Name: store.py
# Author: Sylvia Kwakye
# Copyright: Sylvia Kwakye, BMB Labs, Cornell University
#−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

#import numpy
from pylab import load

"""
miniEC_data_template.py
Text and XML templates for a miniEC data file
"""

metadata = {'potential':'',
            'interval':'',
            'duration':'',
            'range':'1250',
            'channels':'',
            'serial_id':'',
            'firmware':'',
            'software':'',
            'time_stamp':'',
            'date_stamp':'',
            'user_id':'',
            'test_label':'',
            'note':''}


chdata = {'time_data':'',
          'ch1_data':'',
          'ch2_data':'',
          'ch3_data':'',
          'ch4_data':'',
          'ch5_data':''}


#text template for a miniEC data file
def miniEC_text_template():
    """#Experiment: DCPA
#Potential [mV]: $potential_here$
#Interval [sec]: $interval_here$
#Duration [min]: $duration_here$
#Range [nA]: $range_here$
#Time: $date_stamp_here$ $time_stamp_here$
#
#User ID: $user_id_here$
#Test Label: $test_label_here$
#Software version: $software_here$
#Firmware version: $firmware_here$
#Serial ID: $serial_id_here$
```

```
#
#Time (s) Chan-1 Chan-2  Chan-3  Chan-4  Chan-5
"""



#XML template for an exported miniEC data file
def miniEC_xml_template():
    """<?xml version="1.0" encoding="UTF-8"?>
<miniec>
    <head>
        <experiment>DCPA</experiment>
        <date_stamp>$date_stamp_here$</date_stamp>
        <time_stamp>$time_stamp_here$</time_stamp>
        <interval units='sec'>$interval_here$</interval>
        <duration units='mins'>$duration_here$</duration>
        <potential units='mV'>$potential_here$</potential>
        <range units='nA'>$range_here$</range>
        <channels>$channels_here$</channels>
    </head>

    <extra>
        <note>$note_here$</note>
        <user>$user_id_here$</user>
        <test_label>$test_label_here$</test_label>
        <software>$software_here$</software>
        <firmware>$firmware_here$</firmware>
        <serial_id>$serial_id_here$</serial_id>
    </extra>

    <ecdata>
        <time units = 'sec'>$time_data_here$</time>
        <channel units = 'nA' number = '1'>$ch1_data_here$</channel>
        <channel units = 'nA' number = '2'>$ch2_data_here$</channel>
        <channel units = 'nA' number = '3'>$ch3_data_here$</channel>
        <channel units = 'nA' number = '4'>$ch4_data_here$</channel>
        <channel units = 'nA' number = '5'>$ch5_data_here$</channel>
    </ecdata>

</miniec>
"""



#parse miniEC header info into the metadata dictionary
def parse_miniEC_metadata(md):
    """ Parse metadata list to metadata """
    metadata['potential'] = md[0]
    metadata['interval'] = md[1]
    metadata['duration'] = md[2]
    metadata['range'] = md[3]
    metadata['channels'] = md[4]
    metadata['serial_id'] = md[5]
    metadata['firmware'] = md[6]
    metadata['software'] = md[7]
    metadata['time_stamp'] = md[8]
```

```
    metadata['date_stamp'] = md[9]
    metadata['test_label'] = md[10]
    metadata['user_id'] = md[11]
    metadata['note'] = md[12]


#parse miniEC data from a text format file
def text_to_xml(filename, newfile):
    filehandle = open(filename, 'r' )

    while True:
        line = filehandle.readline()

        if line.startswith('#Time:'):
            parts = line.split(" ")
            metadata['date_stamp'] = parts[1].strip()
            metadata['time_stamp'] = parts[2].strip()

        else:
            parts = (line.strip()).split(':')
            if parts[0].startswith('#Potential'):
                metadata['potential'] = parts[1]

            if parts[0].startswith('#Interval'):
                metadata['interval'] = parts[1]

            if parts[0].startswith('#Duration'):
                metadata['duration'] = parts[1]

            if parts[0].startswith('#Range'):
                metadata['range'] = parts[1]

            if parts[0].startswith('#Channels'):
                metadata['channels'] = parts[1]

            if parts[0].startswith('#Serial'):
                metadata['serial_id'] = parts[1]

            if parts[0].startswith('#Firmware'):
                metadata['firmware'] = parts[1]

            if parts[0].startswith('#Software'):
                metadata['software'] = parts[1]

            if parts[0].startswith('#User'):
                metadata['user_id'] = parts[1]

            if parts[0].startswith('#Test'):
                metadata['test_label'] = parts[1]

            if parts[0].startswith('#Note'):
                metadata['note'] = parts[1]

            if line.startswith('#') is False:
                break
```

```
                filehandle.close


    #""" Parse recorded data to chdata
    channels = load(filename, comments='#')
    chdata['time_data'] = ' '.join(['num' for num in channels[:,0]])
    chdata['ch1_data'] = ' '.join(['num' for num in channels[:,1]])
    chdata['ch2_data'] = ' '.join(['num' for num in channels[:,2]])
    chdata['ch3_data'] = ' '.join(['num' for num in channels[:,3]])
    chdata['ch4_data'] = ' '.join(['num' for num in channels[:,4]])
    chdata['ch5_data'] = ' '.join(['num' for num in channels[:,5]])

    st = miniEC_xml_template.__doc__
    #fill in metadata
    for (key,value) in metadata.items():
        st = st.replace('$%s_here$'%(key,),value)

    #fill in data
    for (key,value) in chdata.items():
        st = st.replace('$%s_here$'%(key,),value)

    try:
        # open file stream
        file = open(newfile, "w")
        print >>file, st
        file.close()
        #message = "Data exported to " + filename
        print "Data exported to ", filename

    except IOError:
        #message = "There was an error writing to " + newfile
        #sys.exit()

        print "There was an error writing to ", newfile
        pass

    print st



#fill in the header for the text file format
def make_text_header(md):
    header = miniEC_text_template.__doc__

    metadata['potential'] = str(md[0])
    metadata['interval'] = str(md[1])
    metadata['duration'] = str(md[2])
    metadata['range'] = str(md[3])
    metadata['channels'] = str(md[4])
    metadata['serial_id'] = md[5]
    metadata['firmware'] = md[6]
    metadata['software'] = md[7]
    metadata['time_stamp'] = md[8]
    metadata['date_stamp'] = md[9]
    metadata['test_label'] = md[10]
```

```
    metadata['user_id'] = md[11]
    metadata['note'] = md[12]

    for (key,value) in metadata.items():
        header = header.replace('$%s_here$'%(key,),value)

    return header




if __name__ == "__main__":
    print miniEC_text_template.__doc__
    print miniEC_xml_template.__doc__
```

A.3.5 miniEC.dtd

Listing A.18: The document type definition for a miniEC data file

```
<!ELEMENT channel ( #PCDATA ) >
<!ATTLIST channel number NMTOKEN #REQUIRED >
<!ATTLIST channel units NMTOKEN #FIXED "nA" >

<!ELEMENT channels ( #PCDATA ) >

<!ELEMENT date_stamp ( #PCDATA ) >

<!ELEMENT duration ( #PCDATA ) >
<!ATTLIST duration units NMTOKEN #REQUIRED >

<!ELEMENT ecdata ( time, channel+ ) >

<!ELEMENT experiment ( #PCDATA ) >

<!ELEMENT extra ( note, user, test_label, software, firmware, serial_id ) >

<!ELEMENT firmware ( #PCDATA ) >

<!ELEMENT head ( experiment, date_stamp, time_stamp, interval, duration, potential, range, channels ) >

<!ELEMENT interval ( #PCDATA ) >
<!ATTLIST interval units NMTOKEN #REQUIRED >

<!ELEMENT miniec ( head, extra, ecdata ) >

<!ELEMENT note ( #PCDATA ) >

<!ELEMENT potential ( #PCDATA ) >
<!ATTLIST potential units NMTOKEN #REQUIRED >

<!ELEMENT range ( #PCDATA ) >
<!ATTLIST range units NMTOKEN #REQUIRED >

<!ELEMENT serial_id ( #PCDATA ) >

<!ELEMENT software ( #PCDATA ) >

<!ELEMENT test_label ( #PCDATA ) >

<!ELEMENT time ( #PCDATA ) >
<!ATTLIST time units NMTOKEN #REQUIRED >

<!ELEMENT time_stamp ( #PCDATA ) >

<!ELEMENT user ( #PCDATA ) >
```

A.3.6 miniEC.xsd

Listing A.19: The XML schema for a miniEC data file

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="channel">
    <xs:complexType mixed="true">
      <xs:attribute name="units" type="xs:NMTOKEN" use="required" fixed="nA" />
      <xs:attribute name="number" type="xs:positiveInteger" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="channels" type="xs:positiveInteger" />
  <xs:element name="date_stamp" type="xs:string" />

  <xs:element name="duration">
    <xs:complexType mixed="true">
      <xs:attribute name="units" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="ecdata">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="time" minOccurs="1" maxOccurs="1" />
        <xs:element ref="channel" minOccurs="1" maxOccurs="5" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="experiment" type="xs:string" />

  <xs:element name="extra">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="note" />
        <xs:element ref="user" />
        <xs:element ref="test_label" />
        <xs:element ref="software" />
        <xs:element ref="firmware" />
        <xs:element ref="serial_id" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="firmware" type="xs:string" />

  <xs:element name="head">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="experiment" />
        <xs:element ref="date_stamp" />
        <xs:element ref="time_stamp" />
```

```
        <xs:element ref="interval" />
        <xs:element ref="duration" />
        <xs:element ref="potential" />
        <xs:element ref="range" />
        <xs:element ref="channels" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="interval">
    <xs:complexType mixed="true">
      <xs:attribute name="units" type="xs:positiveInteger" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="miniec">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="head" />
        <xs:element ref="extra" />
        <xs:element ref="ecdata" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="note" type="xs:string" />

  <xs:element name="potential">
    <xs:complexType mixed="true">
      <xs:attribute name="units" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="range">
    <xs:complexType mixed="true">
      <xs:attribute name="units" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="serial_id" type="xs:string" />
  <xs:element name="software" type="xs:string" />
  <xs:element name="test_label" type="xs:string" />

  <xs:element name="time">
    <xs:complexType mixed="true">
      <xs:attribute name="units" type="xs:NMTOKEN" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="time_stamp" type="xs:string" />
  <xs:element name="user" type="xs:string" />

</xs:schema>
```

REFERENCES

[1] FEMA, "National situation update: Thursday, May 31, 2007." http://www.fema.gov/emergency/reports/2007/nat053107.shtm, May 2007. Last accessed January 7th 2008.

[2] S. Riley, C. Fraser, C. Donnelly, A. C. Ghani, L. J. Abu-Raddad, A. J. Hedley, G. M. Leung, L. M. Ho, T. H. Lam, T. Q. Thach, P. Chau, K. P. Chan, S. Lo, P. Leung, T. Tsang, W. Ho, K. Lee, E. M. Lau, N. M. Ferguson, and R. M. Anderson, "Transmission dynamics of the etiological agent of SARS in Hong Kong: Impact of public health interventions," *Science* **300**, pp. 961–966, 2003.

[3] WHO, "Coronavirus never before seen in humans is the cause of SARS unprecedented collaboration identifies new pathogen in record time." WHO Programmes and projects, Media centre, News releases 2003, April 16 2003.

[4] WHO, "Epidemiology of WHO-confirmed human cases of avian A(H5N1) infection," *Weekly Epidemiological Record* **81**, pp. 249 – 260, June 2006.

[5] P. Yager, T. Edwards, E. Fu, K. Helton, K. Nelson, M. R. Tam, and B. H. Weigl, "Microfluidic diagnostic technologies for global public health," *Nature* **442**, p. 418, July 2006.

[6] R. Perez-Pena, "Plan could close 20 or more new york hospitals." New York Times, Nov 28, 2006.

[7] "Social trends in america - vol 3::medical infrastructure – hospital closures and access to health care." http://social.jrank.org/pages/1202/Medical-Infrastructure-Hospital-Closures-Access-Health-Care.html. Last accessed November 14th, 2007.

[8] R. M. Andrews, D. E. Stull, I. Fraser, B. Friedman, and R. L. Houchens, "Serving the uninsured: Safety-net hospitals, 2003." http://www.ahrq.gov/data/hcup/factbk8/index.html, January 2007. Last accessed November 26th, 2007.

[9] "Source of tainted spinach finally pinpointed." http://www.msnbc.msn.com/id/17755937/, March 23 2007. Last accessed August 4, 2008.

[10] J. Ritter, "Safety advocates, growers debate produce rules." http://www.usatoday.com/news/health/2006-09-24-produce-rules_x.htm, October 5 2006.

[11] M. Pollan, "The vegetable-industrial complex." New York Times, October 15, 2006.

[12] M. Beretti and D. Stuart, "Food safety and environmental quality impose conflicting demands on Central Coast growers," *California Agriculture* **62**(2), pp. 68 – 73, 2008.

[13] CDC, "Disease listing> foodborne illness." `http://www.cdc.gov/ncidod/dbmd/diseaseinfo/foodborneinfections\_g.htm`, October 2005. Last accessed December 1, 2007.

[14] B. M. Allos, "Association between Campylobacter infection and Guillain-Barr syndrome," *The Journal of infectious diseases* **176 Suppl 2**, pp. S125–S128, Dec 1997.

[15] P. C. Rowe, E. Orrbine, H. Lior, G. A. Wells, and P. N. McLaine, "A prospective study of exposure to verotoxin-producing *escherichia coli* among canadian children with haemolytic uraemic syndrome. the CPKDRC co-investigators.," *Epidemiology and infection* **110**, pp. 1–7, Feb 1993.

[16] C. Ho, M. Itamura, M. Kelley, and R. Hughes, "Review of chemical sensors for in-situ monitoring of volatile contaminants," tech. rep., Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, March 2001.

[17] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler, "Genbank," *Nucleic Acids Research* **35**, pp. D21–D25, Jan 2007.

[18] S. E. Thompson and S. Parthasarathy, "Moore's law: the future of Si microelectronics," *Materials Today* **9**, pp. 20–25, June 2006.

[19] S. K. Sia and G. M. Whitesides, "Microfluidic devices fabricated in poly(dimethylsiloxane) for biological studies," *Electrophoresis* **24**(21), pp. 3563–3576, 2000.

[20] L. Adleman, "Molecular computation of solutions to combinatorial problems," *Science* **266**(5187), pp. 1021 – 1024, 1994.

[21] J. Parker, "Computing with DNA," *EMBO reports* **4**(1), pp. 7–10, 2003.

[22] N. Yachie, K. Sekiyama, J. Sugahara, Y. Ohashi, and M. Tomita,

"Alignment-based approach for durable data storage into living organisms," *Biotechnology Progress* **23**, pp. 501–505, April 2007.

[23] A. Manz, N. Graber, and H. M. Widmer, "Miniaturized total chemical analysis systems: a novel concept for chemical sensing," *Sensors and Actuators B* **B1**, pp. 244–248, 1990.

[24] S. C. Terry, J. H. Jerman, and J. B.Angell, "A gas chromatograph analyzer fabricated on silicon wafer," *IEEE Transactions on Electron Devices* **26**, pp. 1880–1886, 1979.

[25] C.S.Effenhauser, A.Manz, and H.M.Widmer, "Glass chips for high speed capillary electrophoresis separations with sub-micron plate heights," *Analytical Chemistry* **65**, pp. 2637–2642, 1993.

[26] Y. Huang and B. Rubinsky, "Microfabricated electroporation chip for single cell membrane permeabilization," *Sensors and Actuators A* **89**, pp. 242–249, 2001.

[27] B. C. Giordano, J. Ferrence, S. Swedberg, A. F. R. Huhmer, and J. P. Landers, "Polymerase chain reaction in polymeric microchips: DNA amplification in less than 240 seconds," *Analytical Biochemistry* **291**, pp. 124–132, 2001.

[28] M. U. Kopp, A. J. de Mello, and A. Manz, "Chemical amplification: Continuous-flow PCR on a chip," *Science* **280**(5366), p. 1046, 1998.

[29] J. A. Fruetel, R. F. Renzi, V. Vandernoot, J. Stamps, B. A. Horn, J. A. West, S. Ferko, R. Crocker, C. G. Bailey, D. Arnold, B. Wiedenman, W. Y. Choi, D.Yee, I. Shokair, E. Hasselbrink, P. Paul, D. Rakestraw, and D. Padgen, "Microchip separations of protein biotoxins using an integrated hand-held device," *Electrophoresis* **26**(6), pp. 1144–1154, 2005.

[30] C. Papadea, J. Foster, S. Grant, S. A. Ballard, J. C. Cate IV, W. M. Southgate, and D. M. Purohit, "Evaluation of the i-STAT portable clinical analyzer for point-of-care blood testing in the intensive care units of a university children's hospital," *Annals of clinical and laboratory science* **32**, pp. 231 – 243, 2002.

[31] C. K. Ho, A. Robinson, D. R. Miller, and M. J. Davis, "Overview of sensors and needs for environmental monitoring," *Sensors* **5**, pp. 4–37, February 2005.

[32] "Two sandia microchemlab technologies soon to be checking for toxins in the nation's water supplies." http://www.sandia.gov/news-center/news-releases/2005/elect-semi-sensors/water.html, September 2005. Last accessed December 1st 2007.

[33] "I-stat corporation/de . 8-k . for 10/9/02 . ex-99.1." http://www.secinfo.com/dRqWm.31556.d.htm, September 2002. Last accessed November 14th, 2007.

[34] F. S. Apple, M. M. Murakami, R. H. Christenson, J. L. Campbell, C. J. Miller, K. G. Hock, and M. G. Scott, "Analytical performance of the i-STAT cardiac troponin I assay," *Clinica Chimica Acta* **345**, pp. 123–127, Jul 2004.

[35] A. F. Rossi and D. Khan, "Point of care testing: improving pediatric outcomes.," *Clinical biochemistry* **37**, pp. 456–461, Jun 2004.

[36] C. Sachs, P. Finetti, P. Rabouine, and F. Abdoulaye, "Analytical performances of the i-STAT portable clinical analyzer for the measurement of PO2 and PCO2," *Annales de biologie clinique* **60**(4), pp. 411–420, 2002.

[37] A. J. Macnab, G. Grant, K. Stevens, F. Gagnon, R. Noble, and C. Sun, "Cost: benefit of point-of-care blood gas analysis vs. laboratory measurement during stabilization prior to transport.," *Prehospital and disaster medicine* **18**(1), pp. 24–28, 2003.

[38] "Screenshots acquired from ABO card demonstration movie." http://www.micronics.net/products/ABO%20card.wmv, August 2007. Last accessed, December 15, 2008.

[39] B. H. Weigl, "Microfluidics-based enteric pathogen assay for developing country applications." http://images.vertmarkets.com/CRLive/files/downloads/f7f70c44-c7a4-459b-bf0b-40cd358f5218/PATH_presentation.pdf, February 2007. Last accessed December 15 2008.

[40] Y. S. Kim, Y. S. Yang, S.-C. Ha, H.-B. Pyo, and C. A. Choi, "Miniaturized electronic nose system based on a personal digital assistant," *Electronics and Telecommunications Research Institute Journal* **27**, pp. 585–594, October 2005.

[41] W. R. Bandy and J. P. Peeters, "Patent no. US 7,148,803: Radio frequency identification (RFID) based sensor networks," December 2006.

[42] J. Peeters, "Patent no. US7,109,859: Method and appartus for wide area surveillance of a terrorist or personal threat," September 2006.

[43] J. Wiegant, C. C. Wiesmeijer, J. M. Hoovers, E. Schuuring, A. d'Azzo, J. Vrolijk, H. J. Tanke, and A. K. Raap, "Multiple and sensitive fluorescence in situ hybridization with rhodamine-, fluorescein-, and coumarin-labeled DNAs," *Cytogenetics and cell genetics* **63**(1), pp. 73–76, 1993.

[44] J. H. Kenten, S. Gudibande, J. Link, J. J. Willey, B. Curfman, E. O. Major, and R. J. Massey, "Improved electrochemiluminescent label for DNA probe assays: rapid quantitative assays of HIV-1 polymerase chain reaction products," *Clinical chemistry* **38**, pp. 873–909, 1992.

[45] F. van De Rijke, H. Zijlmans, S. Li, T. Vail, A. K. Raap, R. S. Niedbala, and H. J. Tanke, "Up-converting phosphor reporters for nucleic acid microarrays," *Nature Biotechnology* **19**, pp. 273–276, Mar 2001.

[46] C. Y. Zhang, H. C. Yeh, M. T. Kuroki, and T. H. Wang, "Single-quantum-dot-based DNA nanosensor," *Nature Materials* **4**, pp. 826–831, Nov 2005.

[47] S. Tomlinson, A. Lyga, E. Huguenel, and N. Dattagupta, "Detection of biotinylated nucleic acid hybrids by antibody-coated gold colloid," *Analytical Biochemistry* **171**(1), pp. 217–222, 1988.

[48] C. Zhang, J. Xu, W. Ma, and W. Zheng, "PCR microfluidic devices for DNA amplification," *Biotechnology Advances* **24**(3), pp. 243–284, 2006.

[49] H. Li and L. Rothberg, "Colorimetric detection of DNA sequences based on electrostatic interactions with unmodified gold nanoparticles," *Proc. Natl. Acad. Sci. USA* **101**, pp. 14036–14039, Sep 2004.

[50] G. S. Rule, R. A. Montagna, and R. A. Durst, "Rapid method for visual identification of specific DNA sequences based on DNA-tagged liposomes," *Clinical chemistry* **42**(8), pp. 1206–1209, 1996.

[51] M. B. Esch, A. J. Baeumner, and R. A. Durst, "Detection of viable *Cryptosporidium parvum* on using oligonucleotide-tagged liposomes in a competitive assay format," *Analytical Chemistry* **73**, pp. 3162–3167, 2001.

[52] J. D. Watson, M. Gilman, J. Witkowski, and M. Zoller, *Recombinant DNA*, Scientific American Books, W. H. Freeman and Company, New York., second ed., 1992.

[53] S. Lee, H. Yowanto, and Y. Tai, "A micro cell lysis device," in *Proc. Eleventh Annual International Workshop on Micro Electro Mechanical Systems MEMS 98*, pp. 443–447, 1998.

[54] S. Branger, J. P. Casalta, G. Habib, F. Collard, and D. Raoult, "*Streptococcus pneumoniae* endocarditis: persistence of DNA on heart valve material 7 years after infectious episode," *Journal of clinical microbiology* **41**, pp. 4435–4437, 2003.

[55] G. M. van der Vliet, P. Schepers, R. A. Schukkink, B. van Gemen, and P. R. Klatser, "Assessment of mycobacterial viability by RNA amplification," *Antimicrobial Agents and Chemotherapy* **38**, pp. 1959–1965, Sep 1994.

[56] T. J. Hellyer, L. E. DesJardin, G. L. Hehman, M. D. Cave, and K. D. Eisenach, "Quantitative analysis of mRNA as a marker for viability of mycobacterium tuberculosis," *Journal of clinical microbiology* **37**, pp. 290–295, 1999.

[57] M. Sela, C. B. Anfinsen, and W. F. Harrington, "The correlation of ribonuclease activity with specific aspects of tertiary structure," *Biochimica et Biophysica Acta* **26**, p. 502, 1957.

[58] A. J. Carpousis, "The E*scherichia coli* RNA degradosome: structure, function and relationship in other ribonucleolytic multienzyme complexes," *Biochemical Society Transactions* **30**, pp. 150–155, 2002.

[59] A. Villarino, O. M. Bouvet, B. Regnault, S. Martin-Delautre, and P. A. D. Grimont, "Exploring the frontier between life and death in E*scherichia coli*: evaluation of different viability markers in live and heat- or uv-killed cells," *Research in microbiology* **151**, pp. 755–768, 2000.

[60] N. Suresh, J. Arora, H. Pant, T. Rana, and U. B. Singh, "Spoligotyping of mycobacterium tuberculosis DNA from archival Ziehl-Neelsen-stained sputum smears," *Journal of Microbiological Methods* **68**(2), pp. 291–295, 2007.

[61] I. Marota, C. Basile, M. Ubaldi, and F. Rollo, "DNA decay rate in papyri and human remains from Egyptian archaeological sites," *American journal of physical anthropology* **121**, pp. 109–111, Jun 2003.

[62] A. M. Vandamme, S. V. Dooren, W. Kok, P. Goubau, K. Fransen, T. Kievits, J. C. Schmit, E. D. Clercq, and J. Desmyter, "Detection of HIV-1 RNA in plasma and serum samples using the NASBA amplification system compared to RNA-PCR," *Journal of Virological Methods* **52**, pp. 121–32, Mar 1995.

[63] V. Phongsisay, V. N. Perera, and B. N. Fry, "Evaluation of eight RNA isolation methods for transcriptional analysis in C*ampylobacter jejuni*," *Journal of Microbiological Methods* **68**, pp. 427–429, Feb 2007.

[64] B. Deiman, P. van Aarle, and P. Sillekins, "Characteristics and applications of nucleic acid sequence-based amplification (NASBA)," *Molecular Biotechnology* **20**, pp. 163–179, 2002.

[65] J. Compton, "Nucleic acid sequence-based amplification," *Nature* **350**, pp. 91–92, 1991.

[66] F. Lunel, M. Mariotti, P. Cresta, I. D. L. Croix, J. M. Huraux, and J. Lefrere, "Comparative study of conventional and novel strategies for the detection of hepatitis C virus RNA in serum: Amplicor, branched DNA, NASBA and in-house PCR," *Journal of Virological Methods* **54**, pp. 159–171, 1995.

[67] R. M. Rocco, ed., *Landmark papers in clinical chemistry*, Elsevier Science & Technology, San Diego, CA, 1 ed., November 2005.

[68] C. Glad and A. Grubb, "Immunocapillarymigration–a new method for immunochemical quantitation," *Analytical Biochemistry* **85**, pp. 180–187, March 1978.

[69] T. R. Peredy and R. D. Powers, "Bedside diagnostic testing of body fluids.," *The American journal of emergency medicine* **15**, pp. 400–407, Jul 1997.

[70] C. F. Aldus, A. V. Amerongen, R. M. C. Arins, M. W. Peck, J. H. Wichers, and G. M. Wyatt, "Principles of some novel rapid dipstick methods for detection and characterization of verotoxigenic E*scherichia coli*," *Journal of applied microbiology* **95**(2), pp. 380–389, 2003.

[71] M. S. Kim and M. P. Doyle, "Dipstick immunoassay to detect enterohemorrhagic E*scherichia coli* O157:H7 in retail ground beef.," *Applied and Environmental Microbiology* **58**, pp. 1764–1767, May 1992.

[72] C. Morissette, J. Goulet, and G. Lamoureux, "Rapid and sensitive sandwich enzyme-linked immunosorbent assay for detection of staphylococcal enterotoxin B in cheese.," *Applied and Environmental Microbiology* **57**, pp. 836–842, Mar 1991.

[73] C. Wittmann, U. Bilitewski, T. Giersch, U. Kettling, and R. D. Schmid, "Development and evaluation of a dipstick immunoassay format for the de-

termination of atrazine residues on-site.," *Analyst* **121**, pp. 863–869, Jun 1996.

[74] J. A. Gabaldn, J. M. Cascales, S. Morias, A. Maquieira, and R. Puchades, "Determination of atrazine and carbaryl pesticide residues in vegetable samples using a multianalyte dipstick immunoassay format.," *Food Additives & Contaminants* **20**, pp. 707–715, Aug 2003.

[75] C. L. Schauer, M.-S. Chen, R. R. Price, P. E. Schoen, and F. S. Ligler, "Colored thin films for specific metal ion detection.," *Environmental Science and Technology* **38**, pp. 4409–4413, Aug 2004.

[76] K. M. Bosompem, I. Ayi, W. K. Anyan, F. K. Nkrumah, and S. Kojima, "Limited field evaluation of a rapid monoclonal antibody-based dipstick assay for urinary schistosomiasis," *Hybridoma* **15**, pp. 443–447, Dec 1996.

[77] K. M. Bosompem, I. Ayi, W. K. Anyan, T. Arishima, F. K. Nkrumah, and S. Kojima, "A monoclonal antibody-based dipstick assay for diagnosis of urinary schistosomiasis.," *Transactions of the Royal Society of Tropical Medicine and Hygiene* **91**(5), pp. 554–556, 1997.

[78] M. Cayemittes, C. Hankins, and M. R. Tam, "An AIDS test that travels well.," *The International Development Research Centre reports* **21**, pp. 27–28, Jul 1993.

[79] S. L. McKenna, G. K. Muyinda, D. Roth, M. Mwali, N. Ng'andu, A. Myrick, C. Luo, F. H. Priddy, V. M. Hall, A. A. von Lieven, J. R. Sabatino, K. Mark, and S. A. Allen, "Rapid HIV testing and counseling for voluntary testing centers in Africa.," *AIDS* **11 Suppl 1**, pp. S103–S110, Sep 1997.

[80] F. Ketema, C. Zeh, D. C. Edelman, R. Saville, and N. T. Constantine, "Assessment of the performance of a rapid, lateral flow assay for the detection of antibodies to HIV.," *Journal of acquired immune deficiency syndromes* **27**, pp. 63–70, May 2001.

[81] T. Banchongaksorn, S. Prajakwong, W. Rooney, and P. Vickers, "Operational trial of ParaSight-F (dipstick) in the diagnosis of falciparum malaria at the primary health care level.," *The Southeast Asian journal of tropical medicine and public health* **28**, pp. 243–246, Jun 1997.

[82] P. E. Avila, K. Kirchgatter, K. C. S. Brunialti, A. M. Oliveira, R. F. Siciliano, and S. M. D. Santi, "Evaluation of a rapid dipstick test, Malar-Check, for the

diagnosis of P*lasmodium falciparum* malaria in Brazil," *Revista do Instituto de Medicina Tropical de So Paulo* **44**(5), pp. 293–296, 2002.

[83] M. L. Cunha, F. Piovesan-Alves, and L. W. Pang, "Community-based program for malaria case management in the Brazilian Amazon.," *The American journal of tropical medicine and hygiene* **65**, pp. 872–876, Dec 2001.

[84] M. V. Cardinal, R. Reithinger, and R. E. Grtler, "Use of an immunochromatographic dipstick test for rapid detection of T*rypanosoma cruzi* in sera from animal reservoir hosts.," *Journal of clinical microbiology* **44**, pp. 3005–3007, Aug 2006.

[85] O. Delgado, M. D. Feliciangeli, V. Coraspe, S. Silva, A. Perez, and J. Arias, "Value of a dipstick based on recombinant RK39 antigen for differential diagnosis of American visceral leishmaniasis from other sympatric endemic diseases in Venezuela," *Parasite* **8**, pp. 355–357, Dec 2001.

[86] S. Buhrer-Sekula, E. N. Sarno, L. Oskam, S. Koop, I. Wichers, J. A. Nery, L. M. Vieira, H. J. de Matos, W. R. Faber, and P. R. Klatser, "Use of ML dipstick as a tool to classify leprosy patients.," *International journal of leprosy and other mycobacterial diseases* **68**, pp. 456–463, Dec 2000.

[87] K. M. Bosompem, I. A. Bentum, J. Otchere, W. K. Anyan, C. A. Brown, Y. Osada, S. Takeo, S. Kojima, and N. Ohta, "Infant schistosomiasis in Ghana: a survey in an irrigation community.," *Tropical medicine & international health* **9**, pp. 917–922, Aug 2004.

[88] P. K. Dash, M. M. Parida, P. Saxena, A. Abhyankar, C. P. Singh, K. N. Tewari, A. M. Jana, K. Sekhar, and P. V. L. Rao, "Reemergence of dengue virus type-3 (subtype-iii) in india: implications for increased incidence of DHF & DSS," *Virology journal* **3**, p. 55, 2006.

[89] A. J. Baeumner, R. N. Cohen, V. Miksic, and J. Min, "RNA biosensor for the rapid detection of viable E*scherichia coli* in drinking water," *Biosensors and Bioelectronics* **18**(4), pp. 405–413, 2003.

[90] A. Baeumner, J. Pretz, and S. Fang, "A universal nucleic acid sequence biosensor with nanomolar detection limits," *Analytical Chemistry* **76**(4), pp. 888 – 894, 2004.

[91] H. Hartley and A. Baeumner, "Biosensor for the specific detection of a single viable B. *anthracis* spore," *Analyt and Bioanalyt Chem* **376**(3), pp. 319 – 327, 2003.

[92] A. J. Baeumner, C. Jones, C. Wong, and A. Price, "A generic sandwich-type biosensor with nanomolar detection limits," *Analytical and Bioanalytical Chemistry* **378**(6), pp. 1587–1593, 2004.

[93] S. R. Nugen, B. Leonard, and A. J. Baeumner, "Application of a unique server-based oligonucleotide probe selection tool toward a novel biosensor for the detection of S*treptococcus pyogenes*," *Biosensors and Bioelectronics* **22**, pp. 2442–2448, May 2007.

[94] A. Humar, C. Ohrt, M. A. Harrington, D. Pillai, and K. C. Kain, "Parasight F test compared with the polymerase chain reaction and microscopy for the diagnosis of P*lasmodium falciparum* malaria in travelers.," *The American journal of tropical medicine and hygiene* **56**, pp. 44–48, Jan 1997.

[95] S. Kwakye, "A micro-total analysis system based on nucleic acid sequence recognition," Master's thesis, Cornell University, January 2002.

[96] S. Kwakye, "A microfluidic biosensor based on nucleic acid sequence recognition," *Analytical and Bioanalytical Chemistry* **376**(7), pp. 1062–1068, 2003.

[97] N. V. Zaytseva, V. N. Goral, R. A. Montagna, and A. J. Baeumner, "Development of a microfluidic biosensor module for pathogen detection," *Lab on a Chip* **5**, pp. 805–811, 2005.

[98] N. V. Zaytseva, R. A. Montagna, and A. J. Baeumner, "Microfluidic biosensor for the serotype-specific detection of dengue virus," *Analytical Chemistry* **77**, pp. 7520–7527, 2005.

[99] I. Fritsch and Z. P. Aguilar, "Advantages of downsizing electrochemical detection for DNA assays.," *Analytical and Bioanalytical Chemistry* **387**, pp. 159–163, Jan 2007.

[100] T. G. Drummond, M. G. Hill, and J. K. Barton, "Electrochemical DNA sensors.," *Nature Biotechnology* **21**, pp. 1192–1199, Oct 2003.

[101] R. Moeller and W. Fritzsche, "Chip-based electrical detection of DNA," *IEE proceedings. Nanobiotechnology* **152**, pp. 47–51, Feb 2005.

[102] K. J. Odenthal and J. J. Gooding, "An introduction to electrochemical DNA biosensors.," *Analyst* **132**, pp. 603–610, Jul 2007.

[103] J. Wang, G. Rivas, J. Fernandes, J. Lopez, M. Jiang, and R. Waymire,

"Indicator-free electrochemical DNA hybridization biosensor," *Analytica chimica acta* **375**, p. 197, 1998.

[104] J. Wang, "Electrochemical nucleic acid biosensors," *Analytica Chimica Acta* **469**, pp. 63–71, 2002.

[105] K. M. Millan and S. R. Mikkelsen, "Sequence-selective biosensor for DNA based on electroactive hybridization indicators.," *Analytical Chemistry* **65**, pp. 2317–2323, Sep 1993.

[106] M. Nakayama, T. Ihara, K. Nakano, and M. Maeda, "DNA sensors using a ferrocene-oligonucleotide conjugate.," *Talanta* **56**, pp. 857–866, Apr 2002.

[107] C. Xu, H. Cai, P. He, and Y. Fang, "Electrochemical detection of sequence-specific DNA using a DNA probe labeled with aminoferrocene and chitosan modified electrode immobilized with ssDNA.," *Analyst* **126**, pp. 62–65, Jan 2001.

[108] H. Cai, Y. Xu, N. Zhu, P. He, and Y. Fang, "An electrochemical DNA hybridization detection assay based on a silver nanoparticle label.," *Analyst* **127**, pp. 803–808, Jun 2002.

[109] M. Ozsoz, A. Erdem, K. Kerman, D. Ozkan, B. Tugrul, N. Topcuoglu, H. Ekren, and M. Taylan, "Electrochemical genosensor based on colloidal gold nanoparticles for the detection of factor v leiden mutation using disposable pencil graphite electrodes.," *Analytical Chemistry* **75**, pp. 2181–2187, May 2003.

[110] J. Wang, A. N. Kawde, A. Erdem, and M. Salazar, "Magnetic bead-based label-free electrochemical detection of dna hybridization.," *Analyst* **126**, pp. 2020–2024, Nov 2001.

[111] L. Authier, C. Grossiord, and P. Brossier, "Gold nanoparticle-based quantitative electrochemical detection of amplified human cytomegalovirus DNA using disposable microband electrodes," *Analytical Chemistry* **73**, pp. 4450–4456, Sep 2001.

[112] C. N. Campbell, D. Gal, N. Cristler, C. Banditrat, and A. Heller, "Enzyme-amplified amperometric sandwich test for RNA and DNA," *Analytical Chemistry* **74**, pp. 158–162, Jan 2002.

[113] F. Azek, C. Grossiord, M. Joannes, B. Limoges, and P. Brossier, "Hybridiza-

tion assay at a disposable electrochemical biosensor for the attomole detection of amplified human cytomegalovirus DNA," *Analytical Biochemistry* **284**, pp. 107–113, Aug 2000.

[114] L. Tan, Y. Li, T. J. Drake, L. Moroz, K. Wang, J. Li, A. Munteanu, C. J. Yang, K. Martinez, and W. Tan, "Molecular beacons for bioanalytical applications," *The Analyst* **130**, pp. 1002 – 1005, 2005.

[115] Y. Xiao, A. A. Lubin, B. R. Baker, K. W. Plaxco, and A. J. Heeger, "Single-step electronic detection of femtomolar DNA by target-induced strand displacement in an electrode-bound duplex.," *Proceedings of the National Academy of Sciences of the United States of America* **103**, pp. 16677–16680, Nov 2006.

[116] C. Fan, K. W. Plaxco, and A. J. Heeger, "Electrochemical interrogation of conformational changes as a reagentless method for the sequence-specific detection of DNA," *Proceedings of the National Academy of Sciences of the United States of America* **100**, pp. 9134–9137, Aug 2003.

[117] A. B. Steel, T. M. Herne, and M. J. Tarlov, "Electrostatic interactions of redox cations with surface-immobilized and solution dna.," *Bioconjugate chemistry* **10**(3), pp. 419–423, 1999.

[118] S. O. Kelley, E. M. Boon, J. K. Barton, N. M. Jackson, and M. G. Hill, "Single-base mismatch detection based on charge transduction through DNA," *Nucleic Acids Research* **27**, pp. 4830–4837, Dec 1999.

[119] J. H. Min and A. Baeumner, "Characterization and optimization of interdigitated ultramicroelectrode arrays as electrochemical biosensor transducers," *Electroanalysis* **16**(9), pp. 724–729, 2004.

[120] R. H. Liu, J. Yang, R. Lenigk, J. Bonanno, and P. Grodzinski, "Self-contained, fully integrated biochip for sample preparation, polymerase chain reaction amplification, and DNA microarray detection," *Analytical Chemistry* **76**, pp. 1824–1831, 2004.

[121] Z. Trajanoski, P. Wach, and R. Gfrerer, "Portable device for continuous fractionated blood sampling and continuous ex vivo blood glucose monitoring," *Biosensors and Bioelectronics* **11**, pp. 479–487, 1996.

[122] G. Li, N. Z. Ma, and Y. Wang, "A new handheld biosensor for monitoring blood ketones," *Sensors and Actuators B* **109**, pp. 285–290, 2005.

[123] T. Kappes, P. Schnierle, and P. C. Hauser, "Field-portable capillary electrophoresis instrument with potentiometric and amperometric detection," *Analytica chimica acta* **393**, pp. 77–82, 1999.

[124] R. Knake and P. C. Hauser, "Portable instrument for electrochemical gas sensing," *Analytica Chimica Acta* **500**, pp. 145–153, 2003.

[125] M. Bourne, "2005 may be a momentous year for MEMS or maybe just a momentum-building year." Small Times: News About MEMS, Nanotechnology and Microsystems, Jan. 21, 2005, January 2005.

[126] E. Manger, "Trends in electrochemical sensors," *The Analyst* **123**(1998), pp. 1967–1970, 1998.

[127] H. S. Narula and J. G. Harris, "VLSI potentiostat for amperometric measurements for electrolytic reactions," in *Proceedings of the 2004 International Symposium on Circuits and Systems, Vancover, Canada, May 23-26, 2004*, (Vancover, Canada), May 2004.

[128] A. Frey, M. Jenkner, M. Schienle, C. Paulus, B. Holzapfl, P. Schindler-Bauer, F. Hofmann, D. Kuhtmeier, J. Krause, J. Albers, W. Gumbrecht, D. Schmitt-Landiedel, and R. Thewes, "Design of an integrated potentiostat circuit for CMOS biosensor chips," in *Proceedings of the 2003 International Symposium on Circuits and Systems*, May 25-28 2003.

[129] R. Kakerow, H. Kappert, E. Spiegel, and Y. Manoli, "Low-power single-chip CMOS potentiostat," in *Digest of Technical Papers of the 8th International Conference on Solid-State Sensors and Actuators Transducers '95 and Eurosensors IX.*, **1**, pp. 142–145, 1995.

[130] K. Murari, N. Thakor, M. Stanacevic, and G. Cauwenberghs, "Wide-range, picoampere-sensitivity multichannel VLSI potentiostat for neurotransmitter sensing," in *Proceedings of the 26th Annual International Conference*, pp. 1–4, IEEE Engineering in Medicine and Biology Society (EMBS'2004), September 2004.

[131] M. D. Steinberg and C. R. Lowe, "A micropower amperometric potentiostat," *Sensors and Actuators B: Chemical* **97**(2-3), pp. 284–289, 2004.

[132] E. M. Avdikos, M. I. Prodromidis, and C. E. Efstathiou, "Construction and analytical applications of a palm-sized microcontroller-based amperometric analyzer," *Sensors and Actuators B: Chemical* **107**, pp. 372–378, 2005.

[133] S. Kwakye, A. J. Baeumner, and V. N. Goral, "Electrochemical microfluidic biosensor for nucleic acid detection with integrated minipotentiostat," *Biosensors and Bioelectronics* **21**(12), pp. 2217–2223, 2006.

[134] K. Fowler, *Electronic Instrument Design: Architecting for the Life Cycle*, Oxford University Press, New York, New York, third ed., 1996.

[135] M. E. Sandison, N. Anicet, A. Glidle, and J. M. Cooper, "Optimization of the geometry and porosity of microelectrode arrays for sensor design.," *Analytical Chemistry* **74**, pp. 5717–5725, Nov 2002.

[136] A. J. Bard and L. R. Faulkner, *Electrochemical Methods-Fundamentals and Applications*, John Wiley and Sons, New York, 1980.

[137] Y. Iwasaki and M. Morita, "Electrochemical measurements with interdigitated array electrodes," *Current Separations* **14**(1), pp. 2–8, 1995.

[138] V. N. Goral, N. V. Zaytseva, and A. J. Baeumner, "Electrochemical microfluidic biosensor for the detection of nucleic acid sequences.," *Lab on a Chip* **6**, pp. 414–421, Mar 2006.

[139] Maxim, "Application note 882: Rs-232 features explained." `http://www.maxim-ic.com/appnotes.cfm/appnote_number/882`. Last accessed, August 10, 2006.

[140] J. C. Candy and G. C. Temes, *Oversampling Delta-Sigma Data Converters*, IEEE Press, New York, 1991.

[141] D. Duffy, J. McDonald, O. Schueller, and G. Whitesides, "Rapid prototyping of microfluidic systems in poly(dimethylsiloxane)," *Analytical Chemistry* **70**(23), pp. 4974–4984, 1998.

[142] X. Jingdong, L. Locascio, M. Gaitan, and C. Lee, "Room-temperature imprinting method for plastic microchannel fabrication," *Analytical Chemistry* **72**(8), pp. 1930–1933, 2000.

[143] L. Martynova, L. Locascio, M. Gaitan, G. Kramer, R. Christensen, and W. MacCrehan, "Fabrication of plastic microfluid channels by imprinting methods," *Analytical Chemistry* **69**(23), pp. 4783–4789, 1997.

[144] B. H. Weigl, "Microfluidics-based lab-on-a-chip systems," *IVD Technology* **6**, pp. 47–53, Nov-Dec 2000.

[145] V. Linder, S. K. Sia, and G. M. Whitesides, "Reagent-loaded cartridges for valveless and automated fluid delivery in microfluidic devices," *Analytical Chemistry* **77**(1), pp. 64–71, 2005.

[146] B. Zheng and R. F. Ismagilov, "A microfluidic approach for screening submicroliter volumes against multiple reagents by using preformed arrays of nanoliter plugs in a three-phase liquid/liquid/gas flow," *Angewandte Chemie International* **44**(17), pp. 2520–2523, 2005.

[147] K. S. Phillips and Q. Cheng, "Microfluidic immunoassay for bacterial toxins with supported phospholipid bilayer membranes on poly(dimethylsiloxane)," *Analytical Chemistry* **77**(1), pp. 327–334, 2005.

[148] A. Llobera, R. Wilke, and S. Battgenbach, "Poly(dimethylsiloxane) hollow abbe prism with microlenses for detection based on absorption and refractive index shift," *Lab on a Chip* **4**(1), pp. 24–27, 2004.

[149] R. B. Fair, A. Khlystov, V. Srinivasan, V. K. Pamula, and K. N. Weaver, "Integrated chemicalbiochemical sample collection, pre-concentration, and analysis on a digital microfluidic lab-on-a-chip platform," in *Proceedings of SPIE Optics East*, L. A. Smith and D. Sobek, eds., *Lab-on-a-Chip: Platforms, Devices, and Applications, Conf. 5591* **5591**, pp. 113–124, (Philadelphia, PA), October 2004.

[150] G. Monti, L. D. Napoli, P. Mainolfi, R. Barone, M. Guida, G. Marino, and A. Amoresano, "Monitoring food quality by microfluidic electrophoresis, gas chromatography, and mass spectrometry techniques: effects of aquaculture on the sea bass (*dicentrarchus labrax*)," *Analytical Chemistry* **77**, pp. 2587–2594, Apr 2005.

[151] D. Ivnitski, D. J. O'Neil, A. Gattuso, R. Schlicht, M. Calidonna, and R. Fisher, "Nucleic acid approaches for detection and identification of biological warfare and infectious disease agents.," *BioTechniques* **35**, pp. 862–869, Oct 2003.

[152] N. Sergeev, M. Distler, S. Courtney, S. F. Al-Khaldi, D. Volokhov, V. Chizhikov, and A. Rasooly, "Multipathogen oligonucleotide microarray for environmental and biodefense applications," *Biosensors and Bioelectronics* **20**(4), pp. 684–698, 2004.

[153] C. Xi, L. Raskin, and S. A. Boppart, "Evaluation of microfluidic biosensor development using microscopic analysis of molecular beacon hybridization kinetics.," *Biomedical microdevices* **7**, pp. 7–12, Mar 2005.

[154] A. Rothert, S. K. Deo, L. Millner, L. G. Puckett, M. J. Madou, and S. Daunert, "Whole-cell-reporter-gene-based biosensing systems on a compact disk microfluidics platform.," *Analytical Biochemistry* **342**, pp. 11–19, Jul 2005.

[155] A. Baeumner, N. A. Schlesinger, N. S. Slutzki, J. Romano, E. Lee, and R. Montagna, "A biosensor for dengue virus detection: Sensitive, rapid and serotype specific," *Analytical Chemistry* **74**(6), pp. 1442 – 1448, 2002.

[156] M. B. Esch, L. E. Locascio, M. J. Tarlov, and R. A. Durst, "Detection of viable C*ryptosporidium parvum* using DNA-modified liposomes in a microfluidic chip," *Analytical Chemistry* **73**, pp. 2952–2958, 2001.

[157] E. Kress-Rogers, ed., *Handbook of Biosensors and Electronic Noses*, CRC Press, New York, 1997.

[158] S. Hiroaki, "Advances in the microfabrication of electrochemical sensors and systems," *Electroanalysis* **12**(9), pp. 703–715, 2000.

[159] S. Nakamoto, N. Ito, T. Kuriyama, and J. Kimura, "A lift-off method for patterning enzyme-immobilized membranes in multi-biosensors," *Sensors and Actuators* **13**, pp. 165–172, 1988.

[160] J. D. Newman and A. P. F. Turner, "Home blood glucose biosensors: a commercial perspective.," *Biosensors and Bioelectronics* **20**, pp. 2435–2453, Jun 2005.

[161] A. Trauring, "Python: Language of choice for EAI." `http://www.fourm.info/MyArticles/pythonEAI`, January 2003. Last accessed December 15, 2008.

[162] E. P. Kartalov, J. F. Zhong, A. Scherer, S. R. Quake, C. R. Taylor, and W. F. Anderson, "High-throughput multi-antigen microfluidic fluorescence immunoassays.," *Biotechniques* **40**, pp. 85–90, Jan 2006.

[163] J. T. Connelly, S. R. Nugen, W. Borejsza-Wysocki, R. A. Durst, R. A. Montagna, and A. J. Baeumner, "Human pathogenic cryptosporidium species bioanalytical detection method with single oocyst detection capability.," *Anal Bioanal Chem* , Mar 2008.

[164] P. Leijdekkers and V. Gay, "Personal heart monitoring system using smart phones to detect life threatening arrhythmias," in *Proc. 19th IEEE In-*

ternational Symposium on Computer-Based Medical Systems CBMS 2006, pp. 157–164, 2006.

[165] S. Segan, "One razr2, four ways to cut it." PC Magazine, August 13 2007.

[166] J. Donner, "The rules of beeping: Exchanging messages via intentional "missed calls" on mobile phones," *Journal of Computer-Mediated Communication* **13**(1), p. 1, 2007.